R T F M*

Retour
vers les
Techniques Fonctionnelles
et la
Modélisation

Travaux Pratiques

L3 MIAGE

$\mathcal{PL}$

Université Grenoble Alpes
Nicolas.Glade@univ-grenoble-alpes.fr

Ce cours a été rédigé par Nicolas Glade.
La partie XML est très largement inspirée du cours de XML d'Emmanuel Promayon, PhD, enseigné à Polytech'Grenoble
et du cours de Formalisation des données - Technologies XML de Céline Fouard, PhD, et Nicolas Glade, PhD, enseigné en L3 Miage.
Certaines parties sont inspirées de *XML Cours et exercices*, Alexandre Brillant, éditions Eyrolles et *XSLT, Mastering XML Transformations*, Doug Tidwell, O'Reilly editions.
Les références des exercices sont donnés dans le texte.
Si vous souhaitez utiliser ce document, merci de contacter Nicolas.Glade@univ-grenoble-alpes.fr

* RTFM : Read This Fantastic Manual !

Table des matières

Partie A

Ressources

Chapitre I

Ressources en ligne

Vous trouverez ici les ressources nécessaires :

1 Documentations XML/XSD/XSLT/HTML

- Le site W3Schools référence de très nombreux cours et documentations : https://www.w3schools.com/
- HTML : https://www.w3schools.com/html/default.asp
- CSS : https://www.w3schools.com/css/default.asp
- XML/XSD/XSLT/XPath : https://www.w3schools.com/xml/default.asp
- SVG : https://www.w3schools.com/graphics/svg_intro.asp

2 Documentations C# et Java

- La documentation Microsoft C# : https://learn.microsoft.com/fr-fr/dotnet/csharp/
- La documentation C# de W3Schools https://www.w3schools.com/cs/index.php
- La documentation Java Oracle: https://docs.oracle.com/en/java/
- La documentation Java de W3Schools : https://www.w3schools.com/java/default.asp

Chapitre II

Usage des IDE pour XML/XSD/XSLT

Vos IDE (Netbeans ou Jetbrains Rider) disposent d'outils intégrés qui permettent de réaliser des opérations sur des documents XML. Ils permettent notamment de vérifier la conformité d'un document XML, de générer un document XML à partir d'un Schema XML, de vérifier la conformité d'un document par rapport à un schéma, d'appliquer des transformations XSLT.

1 Tester la conformité d'un XML

Tester la conformité d'un document XML, c'est vérifier qu'il respecte bien les règle du XML (cf. le cours).

a) Sous Netbeans. Sous Netbeans, on utilise la petite flèche verte (pas Run !) qui est montrée sur la figure II .1

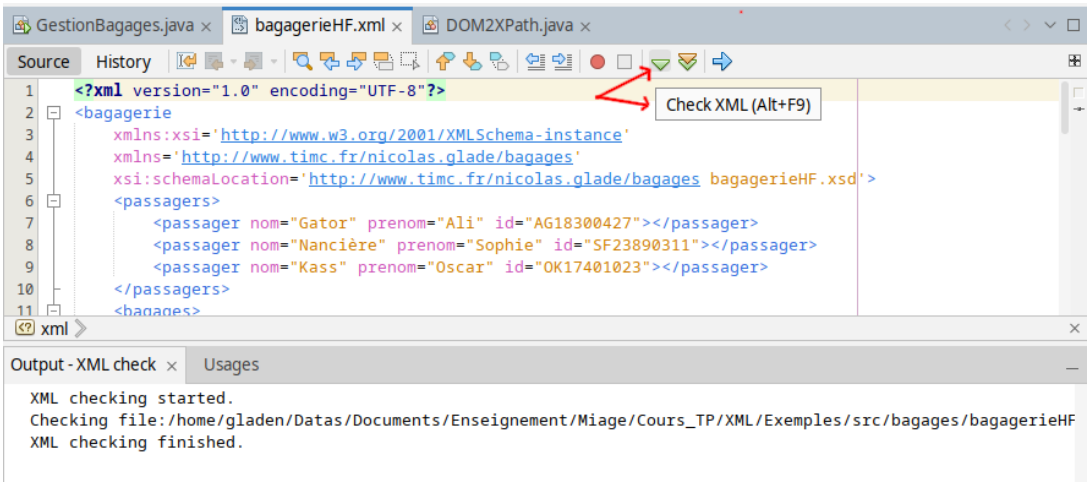

Figure II .1: **Tester la conformité d'un XML sous Netbeans**.

b) Sous Jetbrains Rider. Cet IDE analyse votre code en direct. Les erreurs de conformité sont affichées en rouge et soulignées. De plus, des actions contextuelles peuvent être affichées dans la marge. En laissant la souris sur une erreur, une infobulle apparaît et vous indique quelle est l'erreur.

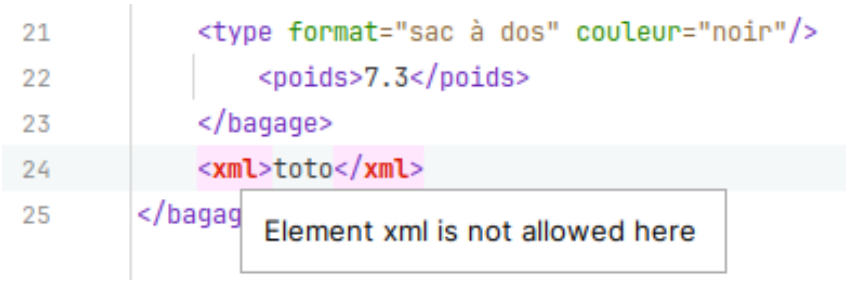

Figure II .2: **Erreur de conformité sous Jetbrain Rider**.

2 Générer un XML à partir d'un Schema

a) Sous Netbeans. Lorsque vous créez un nouveau document XML la fenètre de dialogue vous propose de créer un document contraint par un Schema (voir figure II .3).

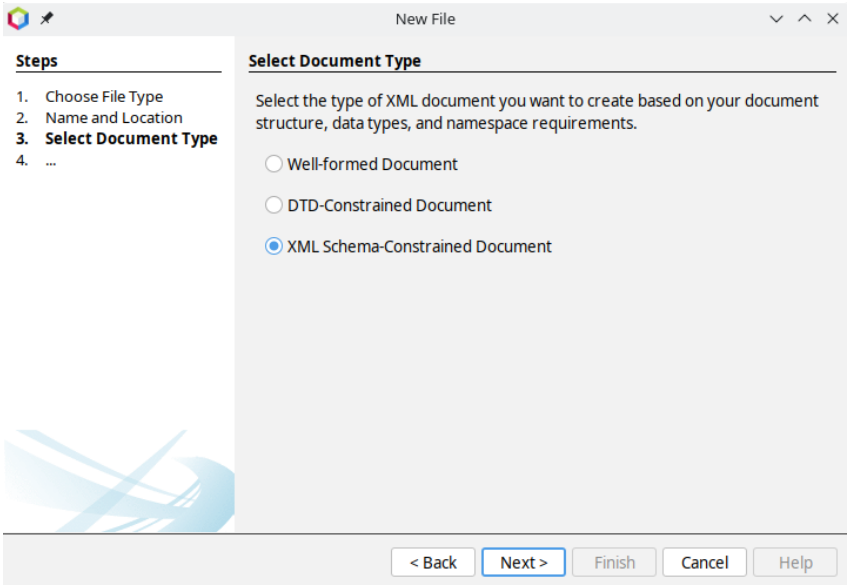

Figure II .3: **Créer une instance XML contrainte sous Netbeans**.

Vous serez alors amené à choisir (par fichier ou namespace) le fichier xsd contenant le schéma que vous voulez utiliser pour contraindre le document XML comme le montre la figure II .4

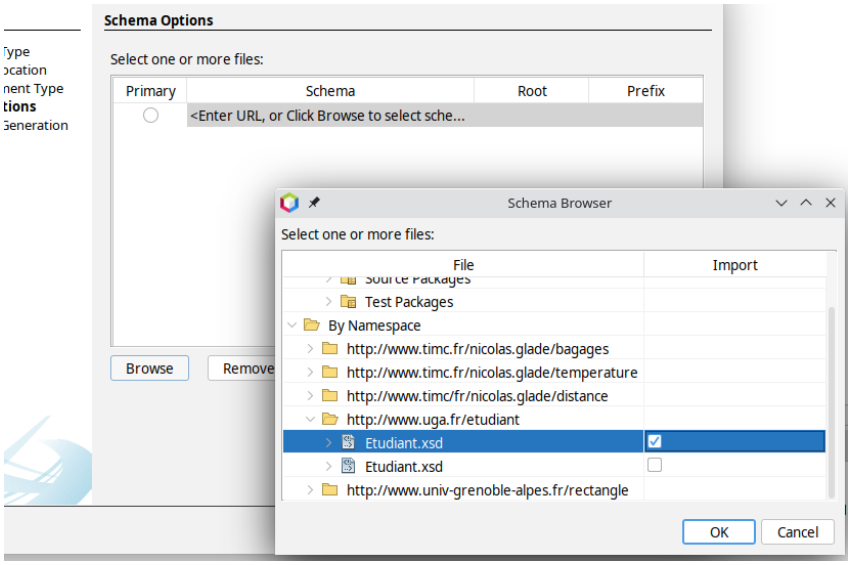

Figure II .4: **Créer une instance XML contrainte sous Netbeans - choix du namespace.**

Enfin (figure II .5), vous devrez choisir l'élément racine parmi ceux disponibles et indiquer un préfixe (par défaut `ns1`, ... pour `namespace 1`, ...). Quand vous rentrez un nouveau préfixe, pensez à bien valider votre saisie avec `entrée` !

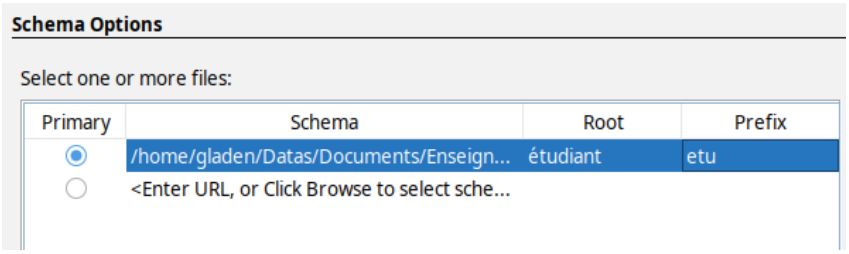

Figure II .5: **Créer une instance XML contrainte sous Netbeans - choix de la racine et du préfixe.**

b) Sous Jetbrains Rider.

- Sélectionnez le Schema XML à partir duquel vous voulez générer une instance de document et ouvrez le. L'onglet actif doit être celui du Schema.
- Dans le menu principal, choisissez `Tools -> XML Actions -> Generate XML Document from XSD Schema`
- Choisissez le schema et donnez le nom que vous voulez à votre instance de document XML
- Dans la liste des éléments (racines) disponibles dans le schéma, sélectionnez celui qui constituera la racine de votre instance.
- Générez

3 Valider un XML par rapport à un Schema

a) Sous Netbeans. Sous Netbeans, on utilise la petite double flêche orange et verte qui est montrée sur la figure II .6

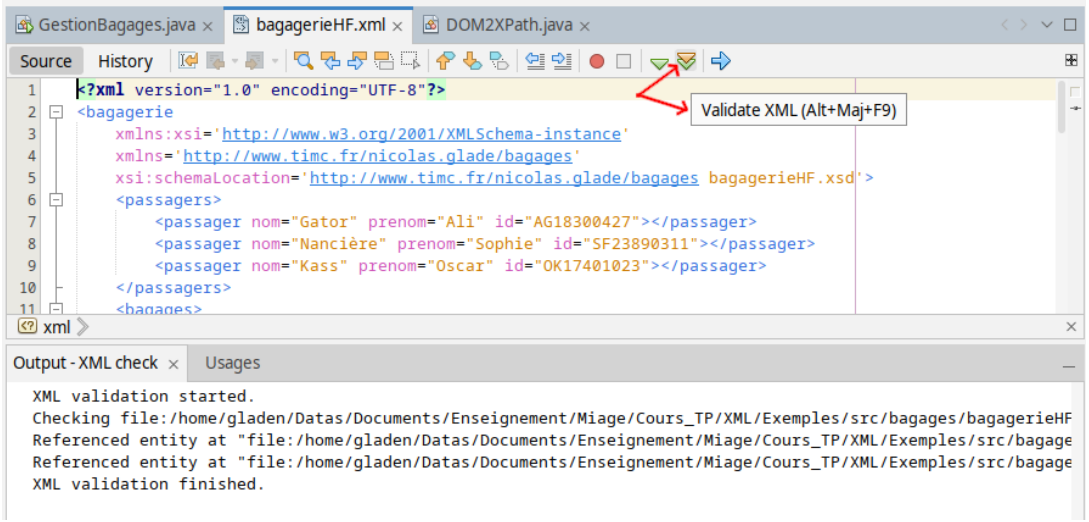

Figure II .6: **Tester la validité d'un XML par rapport à un Schema sous Netbeans**.

b) Sous Jetbrains Rider. Sous Rider comme l'IDE analyse votre code en direct, la plupart des erreurs de validité sont affichées en rouge et une action contextuelle est proposée en marge. Sinon, vous pouvez utiliser l'outil `Validate` depuis le menu contextuel, ou depuis le menu général faire `Tools -> XML Actions -> Validate`.

4 Appliquer une transformation XSLT

a) Sous Netbeans. Avec un clic droit de la souris (menu contextuel) sur le fichier XML dans l'onglet d'exploration ou dans le corps du document, sélectionnez `XSL Transformation...`. Une fenêtre de dialogue s'ouvre (voir figure II .7). Dans cette fenêtre, indiquez le chemin et nom des fichiers XML et XSLT, ainsi que le nom du fichier de sortie (output).

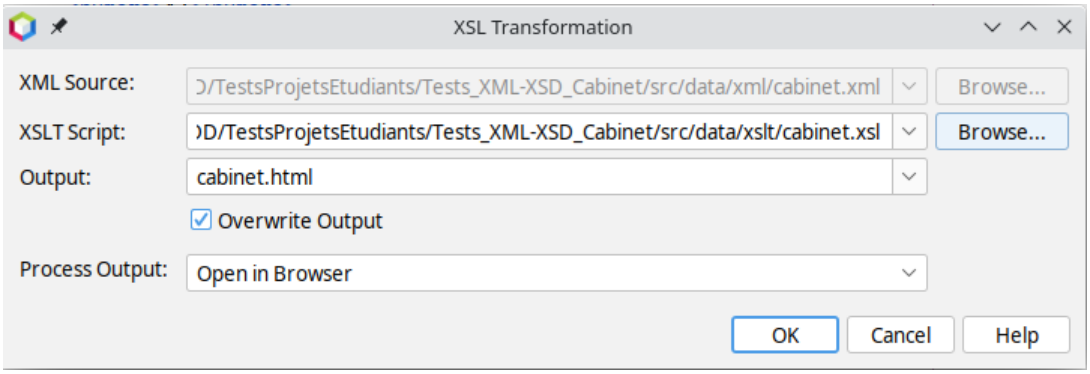

Figure II .7: **Application d'une transformation XSLT à un document XML sous Netbeans**.

b) Sous Jetbrains Rider. Sous Jetbrains, il faut sélectionner le fichier XSL, faire un clic droit à la souris dessus pour faire apparaître le menu contextuel. Dans ce menu, on voit qu'il y a la possibilité de faire `Run 'nomdufichier.xsl'`. Cela ouvre une boite de dialogue permettant d'appliquer la transformation à un fichier XML (voir figure II .8).

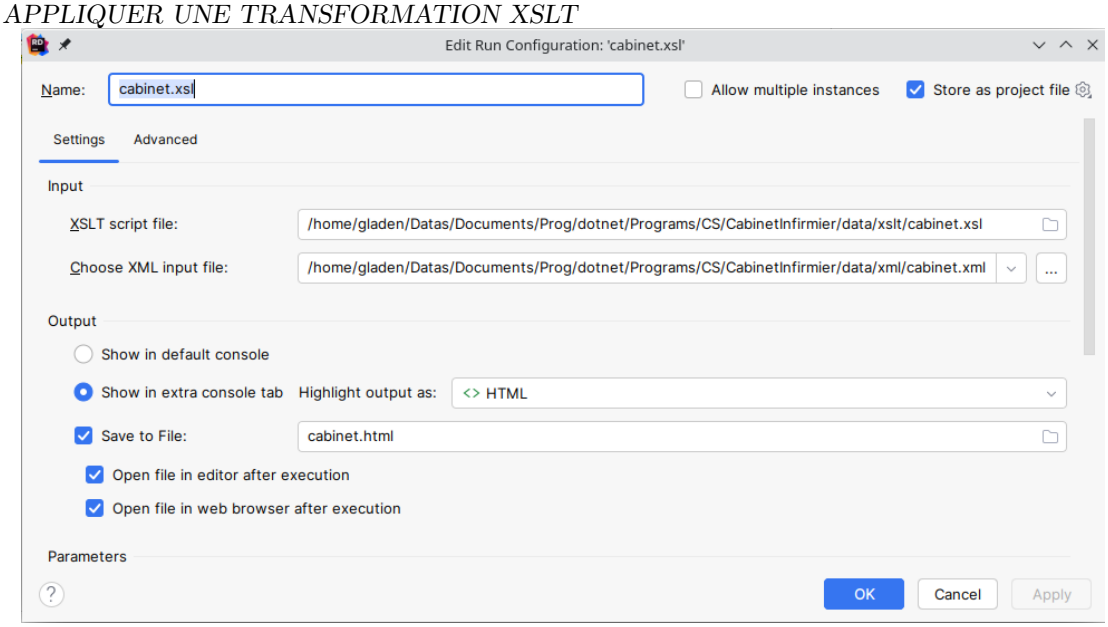

Figure II .8: **Application d'une transformation XSLT à un document XML sous Jetbrains**.

Partie B

Projets

Chapitre III

Projet Cabinet Infirmier

NB : Tous les documents nécessaires à la réalisation de ce projet vous sont fournis sous forme de documents téléchargeables sur la plate-forme.

1 Le cabinet infirmier

1.1 Présentation générale

Le cabinet infirmier *Soins à Grenoble* souhaite organiser les déplacements de ses infirmier(e)s. Il contacte pour cela la société *L3M Agency* pour décider avec elle d'un logiciel capable d'organiser et d'optimiser ses déplacements. Le cabinet emploie aujourd'hui une secrétaire médicale et 3 infirmières, mais est susceptible de s'agrandir. Chaque jour, une infirmière fait le tour de ses patients pour des soins. Chaque jour, la secrétaire médicale entre la liste des patients à visiter ainsi que leurs coordonnées et toute information utile aux soins. Elle affecte ensuite les patients à chacune des infirmières. Lorsqu'une infirmière s'identifie sur l'application, elle obtient la liste des patients qu'elle doit visiter dans la journée, ordonnés de façon à optimiser son trajet quotidien. Elle peut également obtenir la facture correspondant à chaque patient.

1.2 Organisation d'une application exploitant ces données

L'application qui pourrait être développée se présenterait comme un Web Service. Le Web Service permettra aux soignants de se connecter à distance à leur serveur métier pour obtenir divers services, comme la facturation client, et renseignements, comme la liste des patients à visiter ou le trajet optimal pour le visites. Des méthodes d'optimisation d'itinéraire telles que celles qui sont développées en Programmation avec Contraintes et Recherche Opérationnelle (PRCO) et une interface (telle que vous la développeriez en IHM), pouvant se connecter au serveur SOAP, pourraient être utilisés dans le cadre de ce projet, lors de sa finalisation, mais ceci est facultatif.

1.3 Scénario d'utilisation d'une application exploitant ces données

Deux types de clients peuvent s'adresser au serveur: le/la secrétaire médical(e) ou bien un(e) infirmièr(e). L'action du/de la secrétaire médical(e) sera intégralement prise en charge en IHM via le système d'interface. Le serveur métier, lui, va intervenir pour la gestion des requêtes des infirmièr(e)s afin d'obtenir le trajet de la tournée de la journée:

- le serveur reçoit une requête d'un client "infirmière" (via l'IHM)
- le serveur métier fourni la réponse attendue: il lui transmet la requête et le contenu du fichier XML, ou une page HTML
- le module du serveur métier (que vous pourriez développer en FDD-XML) construit une requête à emettre vers GoogleMapAPI pour obtenir les matrices de distances des adresses des patients.

- Le serveur envoie cette requête à GoogleMap API et récupère la réponse sous forme de matrices de distances dans un fichier XML
- le module du serveur métier lit les informations obtenues et appelle les fonctions d'optimisation développées en Recherche Opérationnelle pour calculer le meilleur trajet.
- le serveur métier renvoie ce résultat dans la page résultat de l'application cliente (l'IHM).

2 Objectifs du projet

Dans ce projet, vous allez :

- modéliser la partie données d'un Cabinet Infirmier (UML et XMLSchema)
- développer un programme permettant de vérifier la validité des données XML par rapport au schéma que vous aurez défini
- créer des transformations XSLT permettant d'extraire des connaissances spécifiques (par exemple les données d'un patient particulier) ou de présenter des données sous forme de pages HTML (comme par exemple la liste des interventions d'une infirmière)
- apprendre à utiliser les parseurs et XPath pour extraire des connaissances depuis le fichier XMLSchema
- apprendre à sérialiser ces données
- développer un mini serveur Rest (HTTP) pour accéder à ces données

3 Le document XML

L'application proposée doit stocker les informations des patients et du cabinet. Vous allez donc créer un langage XML pour le stockage de ces informations, c'est à dire modéliser le cabinet XML avec un nouveau Schema XML, à partir des connaissances contenues dans ce document XML.

Il a été convenu que le même document XML stockerait les informations du cabinet (son nom, son adresse et les identifiants des infirmières) ainsi que les données des patients.

3.1 Données du cabinet

Un première partie de ce document XML est représentée dans la figure III .1

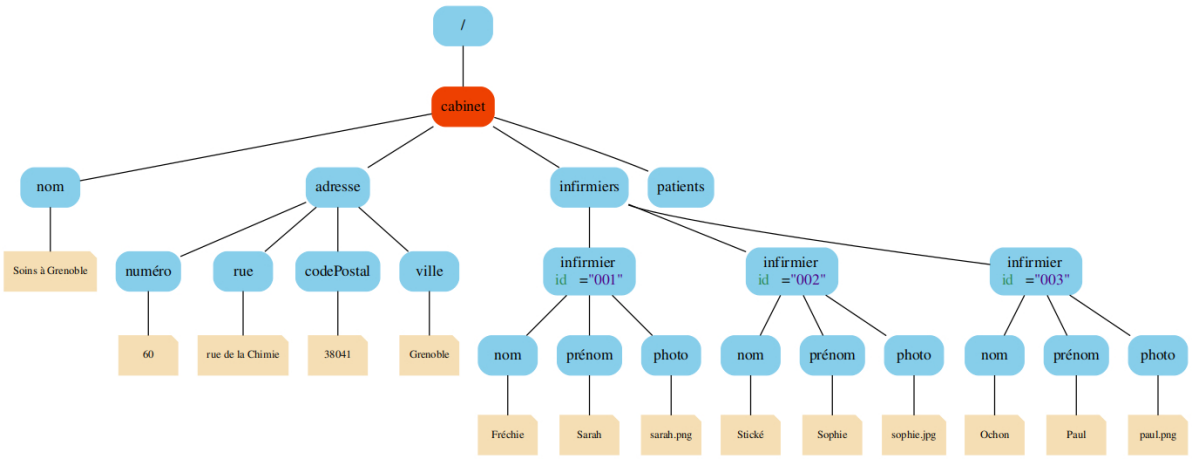

Figure III .1: **Arbre d'instance du cabinet infirmier (données du cabinet)**.

3.2 Données des patients

Après de longues discussions avec le cabinet infirmier et la rédaction du cahier des charges, vous en avez déduit que les informations à rentrer pour chaque patient devait être:

Des données administratives :

- son nom
- son prénom
- sa date de naissance
- son numéro de sécurité sociale
- son adresse précise

Ce document doit aussi comporter, pour chaque patient, et pour chaque visite, sa date, le(s) soin(s) à effectuer par l'infirmière ainsi que son(leur) code NGAP.

Enfin, le(a) secrétaire médical(e) affecte pour chaque patient un(e) infirmièr(e). Cette personne sera représentée par son identifiant (à déterminer).

Un exemple de patient avec toutes les informations précédentes est représenté dans la figure III .2

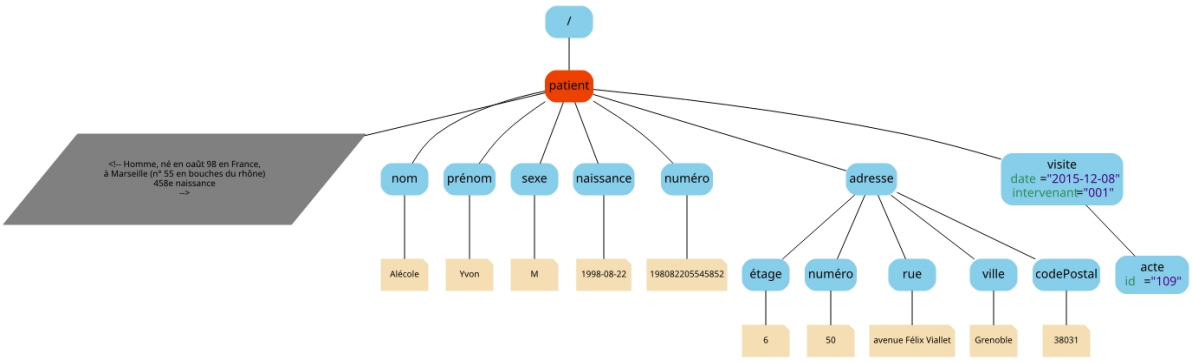

Figure III .2: **Données d'un patient (complément de l'arbre d'instance du cabinet infirmier)**.

3.3 L'adresse

Notez sur le document précédent, que pour que l'adresse du patient soit utilisable par la suite, il faudra bien identifier :

- l'étage (si besoin)
- le numéro de la rue (s'il existe)
- le nom de la rue
- le code postal (un nombre à 5 chiffres)
- la ville

3.4 Actes infirmier et nomenclature NGAP

Revenons à présent sur l'attribut `id` du noeud `acte`. Il correspond à l'identifiant NGAP. Voici un document XML type contenant les actes infirmier et codes NGAP :

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ngap
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns='http://www.univ-grenoble-alpes.fr/l3miage/actes'
    xsi:schemaLocation='http://www.univ-grenoble-alpes.fr/l3miage/actes ../
        schema/actes.xsd'>
    <types>
        <type id="pi">prélèvements et injections</type>
        <type id="pc">pansements courants</type>
        <type id="pl">
            pansements lourds et complexes nécessitant des condition
            d'asepsie rigoueuse
        </type>
        <type id="sd">
            Soins infirmiers à domicile pour un patient, quel que
            soit son âge, en situation de dépendance temporaire ou permanante.
        </type>
    </types>
    <actes>
        <acte id="101" type="pi" clé="AMI" coef="1.5">
            Prélèvement par ponction veineuse directe
        </acte>
        <acte id="102" type="pi" clé="AMI" coef="5">
            Saignée
        </acte>
        <acte id="103" type="pi" clé="AMI" coef="1">
            Prélèvement aseptique cutané ou de sécrétions muqueuses, prélè
                vement
            de selles ou d'urine pour examens sytologiques, bactériologiques,
                mycologiques.
        </acte>
        <!--  ... et plein d'autres actes -->
    </actes>
</ngap>
```

La page suivante explique ce qu'est la nomenclature NGAP pour les soins infirmiers et est à lire attentivement.

Chaque acte est désigné par un intitulé, une lettre-clé et un coefficient.

La lettre-clé est un ensemble de 3 lettres parmi les suivantes:

- AMI pour (Acte Médico-Infirmier)
- AIS (Acte de Soins Infirmier)
- DI (Démarche de soins Infirmiers)

Le coefficient est un chiffre, qui servira de coefficient multiplicateur de la valeur de la lettre-clé pour obtenir la valeur de l'acte. Pour faire simple: Valeur de l'acte = (lettre-clé)xcoefficient.

Vous trouverez la liste complète sur le site de la sécurité sociale, mais nous nous restreindrons à une toute petite partie des actes. Un document plus restreint et plus lisible vous est fourni. Il stocke au format XML les codes NGAP que nous utiliserons dans ce projet. Il sera notamment très utile lorsqu'il vous sera demandé de calculer le coût de chaque visite.

3.5 Numéro de Sécurité Sociale

Si l'on se reporte à Wikipedia, le **numéro de sécurité sociale** en France (nom usuel), ou numéro d'inscription au répertoire des personnes physiques (abrégé en NIRPP ou plus simplement NIR) est un code alphanumérique servant à identifier une personne dans le répertoire national d'identification des personnes physiques (RNIPP) géré par l'Insee. C'est un numéro « signifiant » (c'est-à-dire non aléatoire)

composé de 13 chiffres, suivi d'une clé de contrôle de 2 chiffres.

Pour définir le format du numéro de sécurité sociale, veuillez vous reporter à la documentation Wikipédia. (On pourra éventuellement aussi utiliser ce site).

4 Réalisation - Plan

Les étapes de la réalisation vous sont ici présentées dans l'ordre attendu :

- modélisation UML du cabinet
- modélisation XML Schema du cabinet
- instanciation XML du cabinet
- modélisation de la fiche patient
- transformations XSLT (HTML et XML) du cabinet et du patient
- renforcement des schémas par des contraintes de cohérence
- validation et appels de feuilles de transformation en C#
- usage des parsers (DOM / XmlReader)
- sérialisation
- serveur HTTP

5 Réalisation du projet - 1ere partie (modélisation)

Les étapes de la réalisation vous sont ici présentées dans l'ordre attendu :

- modélisation UML
- génération d'une instance XML
- validation du document
- usage :
  - transformations XSLT
  -

5.1 Découverte du projet

Lisez attentivement l'énoncé... oui celui que vous avez survolé au dessus ! Prenez le temps de bien regarder les arbres d'instance fournis.

5.2 Modélisation UML - Diagramme papier et PlantUML

Dans cette partie, vous allez modéliser sous forme de diagrammes UML le document. Cela requiert une bonne analyse des arbres d'instance.
Vous reviendrez plusieurs fois sur ce travail au cours des différentes séances de TP.
Astuce : pour progresser plus facilement, il est suggéré de modéliser d'abord seulement des sous parties de l'arbre, par exemple une adresse ou un infirmier.

5.2.1 Papier.

Réalisez scrupuleusement toutes ces étapes :

- Commencez par identifier à partir des 2 arbres d'instance précédents les types dont vous avez besoin : listez les.
- Identifiez quels types seront des restrictions (des types simples restreints) et comment.
- Dessinez sur papier un schéma UML contenant tous les types et leurs relations (composition, ...)
- N'oubliez pas d'englober ces types dans un namespace. Le nom du vocabulaire défini pour la base de données du cabinet est `http://www.univ-grenoble-alpes.fr/l3miage/medical`.

5.2.2 UML - PlantUML

Vous allez maintenant implémenter votre diagramme dans le langage de script `PlantUML`. Vous trouverez toute la documentation nécessaire sur ce site, en particulier ici : documentation pour les diagrammes de classe.

Pour tester vos scripts et générer vos diagrammes sous forme d'images, vous devez les soummettre dans le service en ligne PlantUML.

Même remarque que précédemment, progressez doucement par petites implémentations.

5.3 Modélisation XMLSchema

Vous allez écrire le schéma XML implementant le diagramme UML que vous aurez mis au point. Il s'agira de créer un fichier unique nommé (très exactement ; attention au respect de la casse et de l'orthographe) `cabinet.xsd`. Le XMLSchema devra être le plus restrictif possible.

Pour rappel, le nom du vocabulaire défini pour la base de données du cabinet est `http://www.univ-grenoble-alpes.fr/l`

Pensez à vérifier que :

- votre document XML (le schema que vous développez est un document XML) est conforme (voir section 5.5)
- votre document XML (le schema que vous développez est un document XML) est valide par rapport au schéma du vocabulaire `http://www.w3.org/2001/XMLSchema` (voir section 5.6)

5.4 Document XML

5.4.1 Création du document XML

Créez un document xml nommé `cabinet.xml` (très exactement ; attention au respect de la casse et de l'orthographe) et qui représente l'arbre précédent. Ce document devra contenir les données données dans les arbres d'instance.

Idéalement, à ce stade, vous aurez développé préalablement votre Schéma XML... mais il est évident que ce ne sera pas abouti. Vous pouvez donc soit générer l'instance XML à partir du Schema (void chapitre II), soit écrire votre document XML à la main. Dans tous les cas il vous faudra rentrer les données à la main et ... à la fin il faudra que ça marche !

Pensez à vérifier que :

- votre document XML est conforme (voir section 5.5)
- votre document XML est valide par rapport au schéma décrit dans le `cabinet.xsd` (voir section 5.6)

5.4.2 Modifications du document XML

On souhaite ajouter les patients suivant dans le document XML, à la suite du premier patient dans le noeud patients et sur le même modèle.

- Premier patient :
  - Nom: Orouge
  - Prénom: Elvire
  - Elvire est née le 08 mars 82 en France, à Lyon (n° 23 dans le rhône), il s'agissait de la 52e naissance dans cette ville
  - Adresse: Rond-Point de la Croix de Vie 38700 La Tronche.
  - Visite: La visite devra avoir lieu le 08/12/2015 pour un Pansement de brûlure étendue. L'infirmière qui interviendra n'a pas encore été décidée.
- Deuxième patient :
  - Nom: Pien
  - Prénom: Oscare
  - Oscare est née le 25 mars 75 en France, à Chambéry (n° 65 dans la Savoie) Il s'agissait de la 692e naissance.
  - Adresse: rue Casimir Brenier 38000 Grenoble.
  - Visite: La visite devra avoir lieu le 08/12/2015 pour une ponction veineuse directe le et une injection intraveineuse directe isolée d'anthalgiques.
- Troisième patient :
  - Nom: Kapoëtla
  - Prénom: Xavier
  - Xavier est un petit garçon de 4 ans, né le 02 août 2011 en France, à Bordeaux (n° 63 dans la Gironde). Il s'agit de la 35e naissance.
  - Adresse: 25 rue des Martyrs 38042 Grenoble.
  - Visite: La visite devra avoir lieu le 08/12/2015 pour recevoir l'injection de plusieurs allergènes afin d'être désensibilisé.

Ajoutez ces informations. L'objectif ici est que vous compreniez que votre travail va consister non seulement à programmer, mais à vous interesser aux données pour lesquelles vous réalisez ces programmes.

5.5 Conformité de l'instance XML et du XMLSchema

Vous procéderez de 2 manières différentes selon votre avancement dans le projet.

- En premier lieu, vous pouvez utiliser l'outils de vérification de conformité intégré à votre IDE (voir chapitre II)
- Le plus tôt possible, cherchez à utiliser un programme C# pour réaliser cette opération. Tout est décrit dans le cours dans le chapitre sur les parsers.

5.6 Validation de l'instance XML vis-à-vis du XMLSchema

Vous procéderez de 2 manières différentes selon votre avancement dans le projet.

- En premier lieu, vous pouvez utiliser l'outils de validation intégré à votre IDE (voir chapitre II)
- Le plus tôt possible, cherchez à utiliser un programme C# pour réaliser cette opération. Tout est décrit dans le cours dans le chapitre sur les parsers.

5.7 Contraindre davantage : clefs d'existence et d'unicité

Modifiez le Schema XML du centre de soin de façon à ce que les identifiants des médecins soient uniques.

Modifiez le Schema XML du centre de soin de façon à ce que les identifiants des médecins auquels se réfèrent les patients existent.

6 Réalisation du projet - 2ème partie (Transformations XSLT)

6.1 HTML - Le CV des programmeurs

L'objectif ici est une prise en main de HTML, un langage à balise permettant de présenter des documents. Vous avez besoin de savoir écrire du HTML pour ensuite savoir comment transformer vos documents XML en pages HTML.
Dans cet exercice, vous devez tout développer à la main (pas d'outils de génération / design de pages web). Vous n'êtes pas autorisés à utiliser autre chose que HTML et CSS.

Réaliser 2 pages XHTML (une par membre du binôme) qui montrent les CVs des webmasters (vous). Il doit s'agir de votre véritable CV. Ce travail doit être fait consciencieusement. La soin apporté à la page et au contenu sera prise en compte dans l'évaluation du projet.

Ces pages devront contenir chacune au minimum:

- une image (votre photo, par exemple)
- une table (le récapitulatif de votre cursus universitaire par exemple)
- une liste numérotée ou non (par exemple la liste de vos hobbies)
- un lien relatif (vers la page de l'autre webmaster)
- un lien URL vers un site web extérieur

Pour vous aider, vous devez vous appuyer sur la documentation donnée dans le site W3Schools dédié à HTML.

6.2 Transformations XSLT

Dans cette partie, nous allons voir comment automatiser la génération de pages HTML contenant des données extraites de documents XML à l'aide de transformations XSLT. Nous verrons que l'on peut aussi extraire des données pour générer d'autres types de documents, en particulier des documents XML.

6.2.1 La page de l'infirmière

a) ***Contexte.*** Votre application va permettre à un(e) assistant(e) médical(e) d'entrer dans le fichier XML les données du cabinet, des patients et des infirmières.

Dans un deuxième temps, lorsqu'une infirmière s'identifiera sur votre application la requête sera interceptée par un serveur métier (UE ASR) qui appellera une méthode d'optimisation (UE PCRO) vous permettant d'ordonner la liste des visites des patients pour optimiser le trajet de l'infirmière.

Une fois les patients et visites ordonnées pour chaque infirmière, il faudra renvoyer cette information à l'infirmière qui s'est identifiée.

Cet exercice a pour but d'écrire une feuille XSLT qui transforme le document XML contenant les visites triées en une page html qui indique à l'infirmière combien elle a de patients à visiter, et pour chaque patient:

- Son nom
- Son adresse
- La liste des soins à effectuer
- Un bouton qui permettra d'afficher la facture correspondante

Pour afficher une page dédiée à une infirmière en particulier, il faudra passer à la feuille de style un paramètre global. Pour cela, il faut utiliser la ligne suivante, juste avant la définition de la première template (i.e. la template qui match "/").

On pourra choisir une valeur par défaut dans ce paramètre (`001` dans l'exemple ci-dessous) que l'on pourra changer pour tester d'autres infirmières.

```
<xsl:param name="destinedId" select="001"/>
```

b) ***1ere étape : Une première page simple.*** Dans un premier temps, on souhaite que la page n'affiche qu'une phrase du type:
`Bonjour Annie,`

`Aujourd'hui, vous avez 5 patients`

C'est-à-dire que la feuille doit

- Sélectionner le prénom de l'infirmière étant donné son identifiant (donné par le paramètre global de la feuille)
- Compter le nombre de patients pour cette infirmière

Ecrire une feuille XSLT qui, appliquée à votre document XML donne une page HTML contenant les informations ci-dessus.

c) ***2ème étape : inclure la liste des patients et des soins.*** A la suite de la phrase d'accueil, on souhaite lister pour chaque patient à visiter (et dans l'ordre de visite), son nom, son adresse correctement mise en forme et la liste des soins à effectuer. Pour la liste des soins à effectuer, vous devez aller piocher les noms dans le fichier actes.xml (qui vous a déjà été fourni). Pour piocher des informations dans un autre document XML que le document cible de la transformation, on peut par exemple utiliser une variable contenant les noeuds ngap du document actes.xml (placé dans le même répertoire) comme suit:

```
<xsl:variable name="actes" select="document('actes.xml', /)/act:ngap"/>
```

Ecrire les templates nécessaires.

d) ***3ème étape : ajouter un bouton `Facture`.*** Nous allons ajouter, pour chaque patient un bouton Facture qui ouvrira une nouvelle fenêtre avec la facture correspondante.

Ajouter après la liste des soins de chaque patient un bouton HTML qui a pour texte Facture. La documentation du bouton HTML se trouve ici.

Pour générer le bouton via la transformation XSLT, vous devez créer un `xsl:element` nommé `button` et ayant pour texte Facture. Cet élément contiendra un `xsl:attribute` nommé `onclick` comme suit :

```
<xsl:attribute name="onclick">
   openFacture('<xsl:value-of select="??"/>',
               '<xsl:value-of select="??"/>',
               '<xsl:value-of select="??"/>')
</xsl:attribute>
```

Remarque : une autre façon de procéder consiste à créer un élément `input` doté d'attributs `type`, `onclick` et `value` comme expliqué dans la doc HTML.

L'action de ce bouton appelera le script JS `openFacture` donné ci-dessous :

```
function openFacture(prenom, nom, actes) {
       var width  = 500;
       var height = 300;
       if (window.innerWidth) {
           var left = (window.innerWidth-width)/2;
           var top = (window.innerHeight-height)/2;
       } else {
           var left = (document.body.clientWidth-width)/2;
           var top = (document.body.clientHeight-height)/2;
```

```
        }
        var factureWindow = window.open('','facture','menubar=yes, scrollbars=
            yes, top='+top+', left='+left+', width='+width+', height='+height
            +'');
        factureText = "Facture pour : " + prenom + " " + nom;
        factureWindow.document.write(factureText);
    }
```

Ce script devra être intégré dans l'entête du fichier HTML comme expliqué dans la doc.

Lors de l'appel de se script (dans le bouton), les valeurs de nom, prénom et actes seront transmis en paramètres.

Faites en sorte d'appliquer une feuille CSS : Créez une feuille de style CSS pour votre page web html ou complétez celle que vous avez pour votre application, puis ajoutez dans la feuille XSLT l'élément qui permettra à votre feuille html de charger votre feuille de style.

e) ***4ème étape : amélioration de la facture.*** Dans cette partie, nous allons utiliser l'API DOM en Javascript pour afficher la facture d'une visite.
Avant de continuer, vérifiez que l'arborescence de votre projet est la suivante. Tous les fichiers xml, xsd, ... sont dans un dossier data, comme suit :

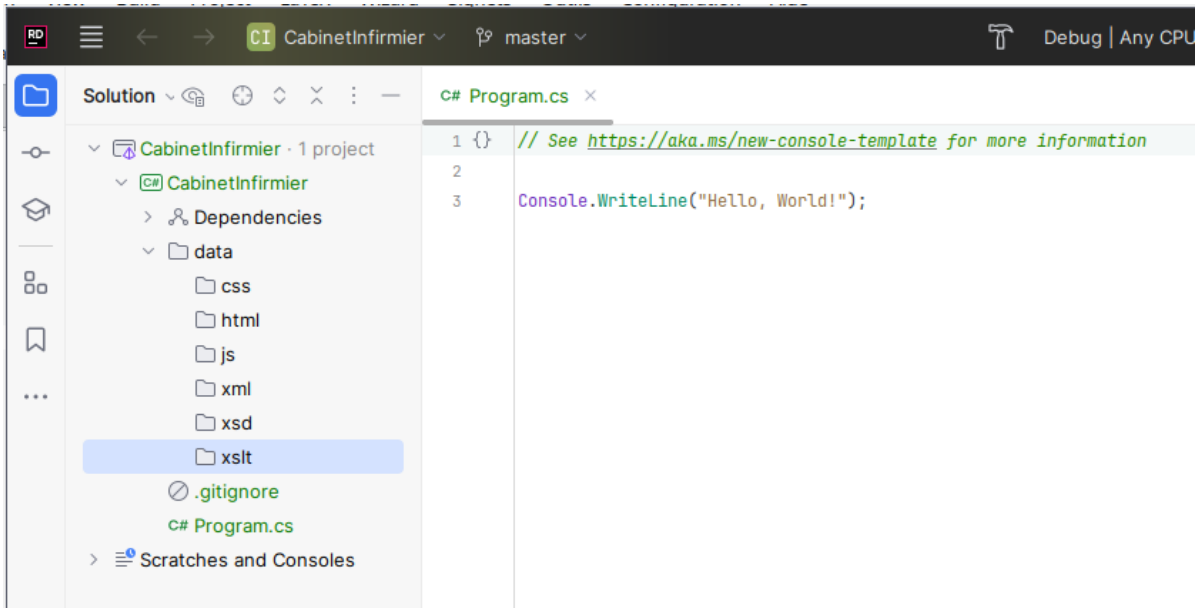

Figure III .3: **Arborescence du projet)**.

Dans le dossier `js`, ajoutez le fichier `facture.js` fourni.

Modifiez la feuille xslt de façon à ce qu'elle appelle ce script comme suit :

```
<script type="text/javascript" src="js/facture.js"></script>
```

Vérifiez son bon fonctionnement.

Modifiez le script pour qu'au lieu d'avoir `var factureText = "Facture pour : " + prenom + " " + nom;` , on ait `var factureText = afficherFacture(prenom, nom, actes);`
Vérifiez son bon fonctionnement.

[Facultatif - à ne faire qu'à la fin du projet] Modifiez/complétez `facture.js` pour que vous ayez une facture correctement mise en page qui affiche:

- l'identité du patient

- son adresse
- son numéro de sécurité sociale
- un tableau qui récapitule les actes, leurs codes et tarif
- la dernière ligne du tableau qui affiche la somme totale à régler

6.2.2 La fiche patient

a) Contexte. Votre application va permettre à un(e) patient(e) de récupérer les informations sur les différents rendez vous qu'il va avoir à partir du fichier XML contenant les données du cabinet, des patients et des infirmières.

Lorsqu'un patient fait cette demande, **un nouveau fichier XML** est généré qui contient les informations le concernant (état civil, adresse, téléphone ...), ce qui lui permettra de vérifier ces informations et d'envoyer une requête de modification le cas échéant. Ce fichier contiendra également la liste des visites médicales le concernant avec toutes les informations écrites en clair, à savoir la date (les visites étant triées par date), le libellé exact de l'acte médical, et le nom de l'infirmier qui viendra pour chaque visite. Vous devez obtenir ce type de document :

```
<?xml version="1.0" encoding="UTF-8"?>
<patient>
    <nom>Pourferlavésel</nom>
    <prénom>Vladimir</prénom>
    <sexe>M</sexe>
    <naissance>1942-12-30</naissance>
    <numéroSS>198082205545842</numéroSS>
    <adresse>
        <rue>Les morets</rue>
        <codePostal>38112</codePostal>
        <ville>Méaudre</ville>
    </adresse>
    <visite date="2017-12-11">
        <intervenant>
            <nom>Fréchie</nom>
            <prénom>Sarah</prénom>
        </intervenant>
        <acte>Ablation de fils ou d'agrafes, plus de dix, y compris le
            pansement éventuel.</acte>
    </visite>
    <visite date="2017-12-08">
        <intervenant>
            <nom>Fréchie</nom>
            <prénom>Sarah</prénom>
        </intervenant>
        <acte>Séance hebdomadaire de surveillance clinique infirmière et de pré
            vention par séance d'une demi-heure.</acte>
    </visite>
</patient>
```

Une fois ce document XML généré et stocké jusqu'à la fin des interventions, un document HTML sera généré grâce à une deuxième feuille de transformation XSLT et renvoyé par le WebService.

b) Identification du patient Pour obtenir ces informations, le patient s'identifiera grâce à son nom. Lorsque le patient se sera connecté au WebService avec son nom. Cette valeur (le nom) sera transmise comme paramètre à la feuille XSLT permettant d'extraire et transformer son dossier en un nouveau document XML.

Dans la première feuille de transformation XLST, comme pour l'infirmière, il faut utiliser la ligne suivante, juste avant la définition de la première template (i.e. la template qui match "/").

On pourra choisir une valeur par défaut dans ce paramètre (Le patient `Pourferlavésel` est demandé dans l'exemple ci-dessous) que l'on pourra changer pour tester d'autres patients.

```
1 <xsl:param name="destinedName" select="Fréchie"/>
```

c) Opérations effectuées sur la fiche patient au cours des TP

- **Première feuille XSLT** :
  - *(Réalisé plus tard)* Fournir au WebService un nom (et un prénom) de patient. Dans l'immédiat, fixer un patient particulier comme indiqué ci-dessus.
  - *(TP d'aujourd'hui)* Modéliser le patient en UML et XML Schema.
  - *(TP d'aujourd'hui)* Ecrire une 1ere feuille XSLT qui utitilise le nom du patient (ex: Pourferlavésel) pour transformer `cabinet.xml` en un nouveau fichier xml nommé `NOMPATIENT.xml` (ex: `Pourferlavésel.xml`) comme demandé. Fixez le nom du fichier. Celui-ci sera plus tard généré par le programme C#.
  - *(TP d'aujourd'hui)* Appliquer la tranformation pour générer le fichier XML patient
- **Deuxième feuille XSLT** :
  (TP d'aujourd'hui) Réaliser une deuxième feuille XSLT qui transforme `NOMPATIENT.xml` en une page html `NOMPATIENT.html` présentant les renseignements voulus.

Pour tester cette feuille vous pouvez modifier votre document XML et affecter des patients à différentes infirmières. Vous pouvez tester divers patients.

7 Réalisation du projet - 3ème partie (Fonctionnalisation)

Dans cette partie, nous allons fonctionnaliser nos données en C#.

Deux remarques :

- Faites bien attention à l'arborescence (précisée à la 4ème étape de la page de l'infirmière). Depuis votre programme, si par exemple un fichier `cabinet.xml` se trouve dans le dossier `data/xml`, le chemin que vous devrez préciser sera `"./data/xml/cabinet.xml"`
- Précisez à la solution dotnet que les fichiers doivent être copiés : par défaut, lors de la construction (build) de la solution (projet) par JetBrains, les fichiers de données (xml, xsd, images ...) ne sont pas copiés dans le dossier `bin/Debug/netX.0/`. Pour qu'ils puissent être copier et que ces fichiers soient disponibles (par exemple `bin/Debug/net8.0/data/xml/cabinet.xml`), il faut le préciser. Pour cela, dans l'arborescence de fichier de JetBrains à gauche (File System), faites un clic droit sur le fichier à ajouter, sélectionnez `Properties`, puis dans la section `Editable`, et dans `Copy to output directory` sélectionnez `Copy always`. (voir par exemple la figure ci-dessous)

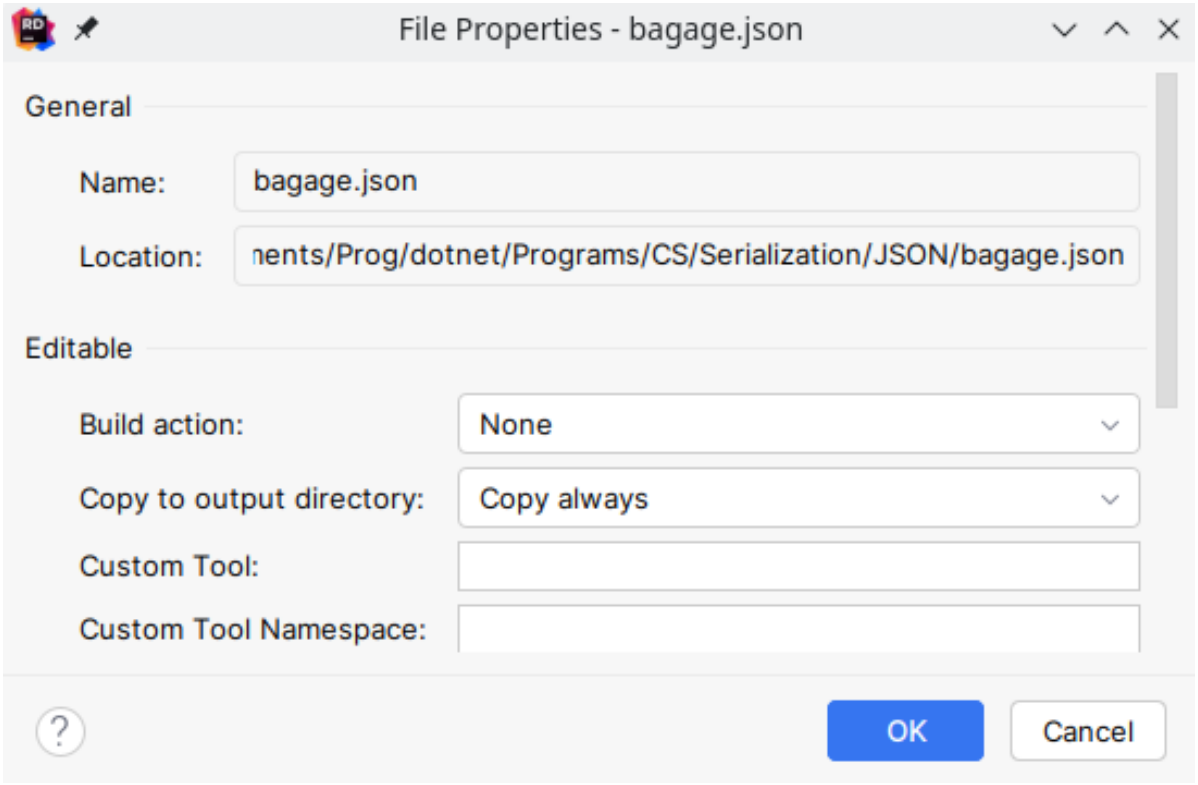

Figure III .4: **Copie d'un fichier pendant la construction).**

7.1 Validation de document XML en C#

Créez une classe statique `XMLUtils`. Cette classe contiendra des méthodes pour travailler sur des documents XML.

Ajoutez-y les méthodes méthodes statique suivantes :

```
1    public static async Task ValidateXmlFileAsync(string schemaNamespace,
         string xsdFilePath, string xmlFilePath) {
2        var settings = new XmlReaderSettings();
3        settings.Schemas.Add(schemaNamespace, xsdFilePath);
4        settings.ValidationType = ValidationType.Schema;
5        Console.WriteLine("Nombre de schemas utilisés dans la validation : "+
             settings.Schemas.Count);
```

```
        settings.ValidationEventHandler += ValidationCallBack;
        var readItems = XmlReader.Create(xmlFilePath, settings);
        while (readItems.Read()) { }
    }


    private static void ValidationCallBack(object? sender, ValidationEventArgs
        e) {
        if (e.Severity.Equals(XmlSeverityType.Warning)) {
            Console.Write("WARNING: ");
            Console.WriteLine(e.Message);
        }
        else if (e.Severity.Equals(XmlSeverityType.Error)) {
            Console.Write("ERROR: ");
            Console.WriteLine(e.Message);
        }
    }
```

Ces méthodes permettent de tester la validation de vos documents XML avec un Schema XML. Remarquez qu'il est indispensable de préciser le namespace. Dans votre programme, vous pouvez appeller cette fonction de validation comme suit :

```
XMLUtils.ValidateXmlFileAsync("http://www.univ-grenoble-alpes.fr/l3miage/
    medical", "./data/xml/cabinet.xml", "./data/xsd/cabinet.xsd");
```

Testez ce programme. Faites de même pour les autres documents et schémas.

Vous penserez plus tard à la possibilité d'inclure cette validation dans vos fonctions (en la modifiant éventuellement).

7.2 Appel de feuilles de transformation XSLT en C#

Dans la classe `XMLUtils` ajoutez la méthode statique suivante :

```
public static void XslTransform(string xmlFilePath, string xsltFilePath, string
     htmlFilePath) {
        XPathDocument xpathDoc = new XPathDocument(xmlFilePath) ;
        XslCompiledTransform xslt = new XslCompiledTransform();
        xslt.Load(xsltFilePath);
        XmlTextWriter htmlWriter = new XmlTextWriter(htmlFilePath,null);
        xslt.Transform(xpathDoc,null,htmlWriter);
    }
```

Utilisez cette fonction pour appliquer les transformations XSLT que vous avez écrites précédemment aux instances XML cibles.

Modifiez votre programme pour qu'il fournisse en paramètre l'identifiant de l'infirmier (ou du patient) à la feuille XSLT. Pour faire cela, suivez la documentation C# ici.

7.3 Parseurs DOM et XmlReader

Ecrivez une classe `Cabinet`. Ecrivez également une classe `Adresse`. Pour l'instant, votre classe `Adresse` ne déclare et instancie qu'un nom et une adresse (pas d'infirmiers ni de patients).

Ces mêmes classes serviront à pratiquer les parsers `XmlReader` et `DOM`, mais aussi, une fois complétées pour correspondre au XML Schema, à pratiquer la sérialisation.

7.3.1 Récupération d'informations à la volée avec XmlReader.

Référez vous au cours pour ajoutez à votre classe `Cabinet` une méthode `void AnalyseGlobale(string filepath)` qui :

- parse un fichier avec un XmlReader

- détecte le type de noeud et l'utilise comme switch
- Affiche un message quand on entre dans le document
- Affiche un message quand on entre dans un élement ; ce message doit afficher le nom de l'élément et le nombre d'attributs qu'il contient
- Affiche un message quand on sort d'un élément
- Affiche un message quand on rencontre du texte (le texte doit être affiché)
- Affiche un message quand on rencontre un attribut (en affichant le nom et le contenu de l'attribut).

Une fois cette fonction testée, sur ce même principe, écrivez une autre fonction qui permet de récupérer le texte d'éléments particuliers (par exemple tous les noms, ou tous les noms des infirmiers). Idéalement, votre recherche doit être passée en paramètre de la fonction.

Créez une autre fonction utilisant le XmlReader pour compter combien d'actes différents devront être effectués, tous patients confondus.

7.3.2 Vérification de présence de valeurs particulières avec DOM.

Apprenez à récupérer des informations particulières en utilisant DOM et XPath en vous inspirant de l'exemple ci-dessous.

```
1 DOM2Xpath bagagesDOM = new DOM2Xpath("./XML/bagagerieHF.xml");
2 String myXpathExpression = "//bg:bagage[bg:description='skis']/@idPassager";
3 XmlNodeList nlBagagesDOM = bagagesDOM.getXPath("bg", "http://www.timc.fr/
      nicolas.glade/bagages", myXpathExpression);
4 foreach (XmlNode node in nlBagagesDOM)
5         Console.WriteLine(node.InnerText);
```

Remarque : que dans votre document vos éléments soient préfixés ou non, vous devez ici préciser un préfixe (même s'il ne correspond pas). Un défaut de l'API `DOM2XPath` oblige à procéder ainsi.

Ecrivez une fonction qui, appliquant un chemin XPath à un document XML, renvoie ainsi une NodeList.

Utilisez ce programme pour vérifier que votre document contient :

- 3 infirmiers
- 4 patients
- une adresse complète pour le cabinet
- une adresse complète pour chaque patient
- qu'un numéro de sécurité sociale est valide par rapport aux informations fournies (date de naissance, et sexe, mais aussi vérifier à l'intérieur du numéro de sécu si la clef est valide).
- que l'ensemble des numéros de sécurité sociale sont valides par rapport aux informations fournies

Une façon de procéder est de créer autant de fonctions que de points à vérifier (en passant les valeurs requises en paramètres), par exemple une méthode `int count(string elementName)` ou une méthode `bool hasAdresse(string elementName)`.

Tout marche bien ? Parfait ! au moins comme ça vous savez comment les profs peuvent évaluer automatiquement vos travaux et pourquoi les noms des éléments et attributs, ainsi que le schéma doivent être strictement conformes au cahier des charges.

7.3.3 Modification de l'arbre DOM et de l'instance XML.

Utilisez DOM pour ajouter un infirmier dont l'id sera 005. Il s'appelle Némard Jean. Vous écrirez une fonction générique d'ajout d'infirmiers qui prend comme paramètre le nom et le prénom de l'infirmier. le nom de sa photo est généré automatiquement (prenom.png) et son identifiant également (ce doit être un incrément par rapport au plus grand id connu de la liste d'infirmiers).

Sur le même principe :

- Utilisez DOM pour ajouter un patient (nom, prénom, adresse, NSS, ...adresse ...). Un patient nouvellement créé n'a aucune visite
- Utilisez DOM pour ajouter des visites à un patient. Une visite doit être affectée à un intervenant.

Ajoutez le patient Mme Niskotch Nicole, dont vous fixerez une date de naissance, lieu de naissance ... pour avoir un NSS valide, à laquelle vous donnerez une adresse. Ajoutez lui ensuite une visite de votre choix avec l'intervenant de votre choix.

7.4 Sérialisation

7.4.1 Sérialisations simples

a) Adresse. Dans un premier temps, écrire un document XML ne contenant qu'une adresse (dont la racine de l'instance XML est une adresse <=> modifiez le schéma XML de façon à permettre la création d'une telle instance). En utilisant ce que vous avez vu en TP sur les XmlSerializer, écrire un programme permettant de sérialiser cette adresse avec un objet de type `Adresse`.

b) Infirmier. Procédez de la même manière pour un `infirmier` en créeant une classe sérialisable `Infirmier`.

c) Modification des classes Adresse et Infirmier. Les champs de ces 2 classes (ex: numéro, rue, codePostal, ...) peuvent être amenés à être modifiés depuis le programme C#, par exemple via un formulaire. Cela signifie que, si aucune précaution n'est prise, une personne mal informée ou mal intentionnée pourrait saisir n'importe quelle valeur (par exemple un code postal abérant comme `38ex79` ou un numéro de rue négatif). En utilisant des propriétés C#, vous pouvez via le setter (`set`) contraindre, en C#, les valeurs à des expressions (un code postal doit correspondre à une expression régulière particulière) ou valeurs spécifiques (un numéro de rue doit être positif et non nul). Modifiez les classes Adresse et Infirmier en leur ajoutant des propriétés (qui deviennent les attributs de classe sérialisés) de telle manière que leurs setters contraignent les valeurs possibles, en accord avec le schéma que vous avez écrit.

d) Questions. Vérifiez que vous savez répondre aux 2 questions suivantes :

- Comment modifier ces classes de telles manière qu'elles ne soient plus modifiables (non mutables) ? Créez une variation immuable des classes `Adresse` et `Infirmier` que vous nommerez `AdresseRO` et `InfirmierRO` (RO pour read only).
- Comment, sans rajouter de propriétés, pourriez vous vous assurer, *a posteriori*, qu'une valeur abérante (non conforme au XML Schema) saisie côté programme C# et sérialisée, puisse quand même être détectée ? Implémentez le !

7.4.2 Sérialisation d'une liste d'infrimiers

Ecrivez une classe sérialisable `Infirmiers` de telle manière qu'elle contienne une liste d'infirmiers. Créez également une instance XML ne contenant que la liste des infirmiers avec pour racine `infirmiers`. Vérifiez que vous êtes bien capables de désérialiser-sérialiser.

7.4.3 Sérialisation du Cabinet

Tout est dans le titre ! Ecrivez toutes les classes manquantes de manière à pouvoir sérialiser/désérialiser le cabinet entier (classe `Cabinet` déjà créée). Faites en sorte de contraindre comme il faut les attributs.

Chapitre IV

Projet Jeu Vidéo

NB : Tous les documents nécessaires à la réalisation de ce projet vous sont fournis sous forme de documents téléchargeables sur la plate-forme.

1 Présentation générale du projet

1.1 Les exigences du projet

Vous allez développer un jeu vidéo en C# en utilisant le framework de jeu vidéo `MonoGame`. Le développement du jeu sera libre. Autrement dit, ce sera a vous de proposer un jeu et d'imaginer un gameplay.

Soyons clairs, vous n'allez pas concevoir et développer le nouveau *Call of Duty*, le nouveau *Skyrim*, ni le nouveau *Mario Kart*. Vous n'en n'aurez pas le temps et ce n'est clairement pas le but.

Le but, je le rappelle est que vous mettiez en pratique des savoir-faire incluant

- la conception d'un cahier des charges
- la structuration de votre code, de vos données
- l'usage de concepts et technologies dont XML/XSD/XSLR, les parsers, la sérialisation, mais aussi la programmation objet, les concepts issus de la programmation fonctionnelle ...

Vous serez notés sur ces critères en très grande partie ; votre implication dans ce projet, le soin que vous apporterez à votre code feront également partie de la notation. Ce projet vous donne donc une grande liberté d'action, sous contrainte d'inclure des éléments technologiques et conceptuels qui seront requis, mais attention, c'est aussi une difficulté car cela va vous demander de vous impliquer fortement dans ce projet et de vous dépasser concernant la programmation. Ce que vous allez réaliser, démarche incluse, préfigure le projet intégrateur qui démarre en décembre.

1.2 Organisation du quadrinome

Vous travaillerez en quadrinomes. L'objectif est que vous participiez tous au code, à la conception ... mais aussi que vous appreniez à vous répartir les tâches (telle partie du code / telle autre partie, ...). Dans tous les cas, vous devez nommer un chef d'équipe qui synchronisera les tâches de votre projet et organisera les points réguliers que vous devez avoir.

1.3 La conception

La conception de votre jeu (son concept) fait partie de l'exercice et le soin qui y sera apporté fera d'une part partie de la notation, d'autre part constituera une étape essentielle de votre travail puisqu'elle vous permettra de guider le développement du programme correspondant.

Vous aurez à :

- Sélectionner ou imaginer un concept de jeu (jeu de rôle, jeu de plate-forme, shooter, puzzle game ...)

- identifier (lister) les composants de base (ex: un joueur, des créatures, une carte positionnant des éléments de décors, ...)
- identifier (lister) les interactions de base (mouvement du joueur, blocages sur la carte (collisions), actions simples du joueur, détection par des créatures, action des créatures ...)
- structurer le code et son fonctionnement via des diagrammes UML. Ces diagrammes sont essentiels. Ils vous permettront d'analyser *a priori* et concevoir l'architecture logicielle de votre jeu. Les deux diagrammes attendus sont :
  - un diagramme de classe : les composants attendus de votre jeu (exemple : créatures, joueur, décors, écran de jeu ...) devront être organisés et devront utiliser intelligemment les notions d'héritage et de composition.
  - le fonctionnement du jeu (déroulement d'une partie, changements de niveaux, actions de création de joueur, de chargement, de sauvergarde ... devront être décrits dans un diagramme fonctionnel.
- apprendre à afficher un objet interactif (ex: player) avec Monogame
- ecrire, par étapes, un programme sans graphiques (sans MonoGame) dans lequel les composants et actions de base son implémentées (sorties console)
- permettre à ces composants de charger des données ou de les sauvegarder (sérialisation, utilisation de parsers)
- ajouter ces éléments (composants et actions) au programme graphique

Il sera demandé de créer une **fiche signalétique du jeu** identique à celles présentées sur ce site dont la lecture détaillée vous donnera tous les éléments pour créer votre jeu : Tutoriel **Ecrire un Game concept**

Mes recommandations personnelles sont :

- Imaginez un jeu simple :
  - simple dans le concept : un jeu compliqué d'un point de vue conceptuel sera difficile à concevoir (à formaliser) puis à implémenter. Un jeu simple peut aboutir à une expérience vidéoludique tout a fait amusante, agréable, intéressante.
  - simple dans la réalisation : ne partez pas dans quelque chose que vous ne pourriez pas réaliser. Par exemple, si vous faites un jeu de rôle, une simple carte limitée à l'écran de jeu, un seul joueur, et quelques interactions suffiront pour prouver le concept de jeu.
- Testez régulièrement des versions fonctionnelles et sauvegardez les.
- Utilisez un dépot git pour ces sauvegardes.

Un jeu simple et un peu moche mais fonctionnel et respectant les exigences de l'UE (les miennes) aura de la valeur. Un jeu compliqué et/ou magnifique mais non fonctionnel et ne respectant pas mes exigences n'aura **aucune** valeur.

1.4 A propos des IA génératrices (chatGPT et ses copains)

Certains d'entre vous vont être très tentés de passer par ces IA génératrices de texte et contenus graphiques. C'est une très mauvaise idée et je vais tout de suite vous mettre en garde :

- d'abord ce ne sera en conséquence pas votre projet mais celui produit par un ChatGPT (ou autre), sachant que ces moteurs fonctionnent en produisant un contenu moyen à partir de ce qui l'a alimenté (d'autres humains). J'ajoute dans ce point que vous auriez peu de fierté à valider un projet / une UE en trichant ... en premier avec vous même, vu que passer par ces systèmes pointera clairement un manque de savoir-faire et/ou d'autonomie.
- et la conséquence du point précédent, c'est que de nos jours on peut sans peine détecter l'usage d'un tel outil (par exemple avec des outils comme ZeroGPT qui ont été entrainés à reconnaitre l'usage de ces IA génératives et qui sont très efficaces).

Autant vous dire que je ne me prive pas de me servir d'outils de détection. Sans compter que nous passerons voir vos projets et verrons dans quelle mesure *vous* vous impliquerez dans vos projets.

2 Quelques concepts de jeux

Ces quelques pitchs pourront vous inspirer. Il est recommandé mais pas obligatoire de suivre un concept voisin d'un de ces 3 jeux.

2.1 Un puzzle d'aventure avec des feux follets

a) Pitch Vous êtes Flamèche, un feu follet. Vous vous êtes égaré dans le bayou. Votre objectif est, de plateau en plateau, d'atteindre à la fin le cimetière dont vous êtes issu pour retrouver votre famille de feux follets. Chaque plateau constitue un niveau avec une entrée (là où vous êtes) et une sortie (un point à atteindre).

Au cours de vos aventures, vous visitez divers biomes (des marais, des forêts humides, des forêts sèches, des prairies, des cours d'eau ...). Plusieurs éléments dans ces milieux sont très combustibles (les branches de bois mort, les arbres morts, les bulles de gaz sortant du marais, les herbes sèches ...), d'autres sont peu combustibles (les branches et arbres verts, les herbes vertes), d'autres sont non combustibles (pierres, sable, terre), d'autres enfin éteignent les feux (chutes et cours d'eau, foret humide ...).

Vous vous déplacez donc de case en case (avec les flèches de direction) mais pouvez rarement vous arrêter autrement que sur les zones non combustibles. Les arrêts même assez courts sur les zones très combustibles mettent le feu. Les arrêts prolongés sur les zones même peu combustibles finissent par déclencher un feu. Les arrêts plus ou moins prolongés sur les zones humides provoquent votre extinction (= votre mort).

Le feu a des effets sur les cases voisines : il se propage. Il a aussi un effet sur vous, s'il vous fait grossir (et vous rend plus dangereux pour l'environnement), il peut être mortel en vous consumant complètement si vous vous y exposez trop longtemps.

Vous comprenez ainsi que les éléments du décors ont leur propre vie : ils ont un état (un niveau de combustibilité positif ou négatif), ils ont une durée d'exposition à votre présence ou a la présence de flammes voisines (augmentation de chaleur = une accumulation de chaleur) ou d'humidité voisine (baisse de chaleur = accumulation de rafraichissement), et ils peuvent changer d'état (passer d'élément naturel à flamme, puis de flamme à tas de cendres non combustible).
Vous même, en tant que feu follet, présentez une durée d'exposition au décors à votre contact : exposition au feu (vous grossissez et augmentez votre action de feu sur les éléments voisins, mais réduisez vos points de vie) ; exposition à l'humidité (vous devenez plus petit et diminuez ainsi votre action calorifique sur les éléments du décors, mais diminuez vos points de vie). Votre vie remonte toute seule progressivement mais lentement.

Le gameplay sera pensé comme un puzzle avec des parties à résoudre pour atteindre vos objectifs à chaque plateau (par exemple traverser une forêt). L'idée est de constituer le gameplay (et de jouer) en exploitant le fait que le joueur puisse volontairement mettre le feu pour provoquer des actions, pour passer, ... ou au contraire puisse volontairement s'humidifier pour éviter de déclencher des incendies.

b) Difficulté : * à **

La réalisation de ce jeu est relativement peu difficile. Les cartes sont simples et sur 1 plateau à chaque fois, les sprites (voir définition plus bas) n'ont pas besoin d'être animés, les actions sont simples à implémenter.

2.2 Le jeu Crazy Classroom : micro-management d'une classe

Sachez que le jeu Crazy Classroom a été réalisé par 2 étudiants de BTS sous ma direction récemment ... et qu'il est plutôt amusant.

a) Pitch Le professeur Krab porte bien son nom puisqu'il passe son temps à se déplacer entre les chaise et les tables comme un crabe pour répondre au questions des étudiants. Pas seulement pour leur répondre d'ailleurs, mais aussi pour leur demander de se taire et de se remettre au travail. Son objectif,

c'est que sa salle de classe reste disciplinée et que les étudiants finissent leur projet.

Les élèves ont un avancement progressif de leur travail (par exemple 1% par seconde de temps réel de jeu), mais cette progression est ponctuée d'arrêts qui mettent en pause l'avancement du travail. Les arrêts sont de 3 nature :

- (A) les questions symbolisées dans une bulle de dialogue verte (un affichage graphique) par un point d'interrogation (**?**).
- (B) les "monsieur/madame, ça marche pas !", des *non-questions* symbolisées dans une bulle de dialogue bleue par un point d'exclamation (**!**).
- (C) le bavardage symbolisé dans une bulle de dialogue rouge par le texte "blablabla".

Ces évènements se produisent de façon probabiliste (tirages aléatoires) avec une fréquence configurable dans un fichier de configuration du jeu. Dans une première version du jeu les évènements sont exclusifs pour chaque élève : si un évènement A, B ou C se déclenche pour un élève, tant qu'il n'est pas débloqué, il ne peut se produire d'autres évènements pour cet élève. Une autre version du jeu peut rendre non exclusif le bavardage par exemple.

Les bulles de dialogue montrant les évènements A, B ou C sont des boutons permettant par un click une ou plusieurs actions de la part du professeur. Dans les 3 situations, le professeur doit se déplacer jusqu'à l'élève pour débloquer la situation.
Parmi les actions possibles, le professeur peut :

- **(A) Question de l'élève** :
  - soit répondre à la question -> l'élève est débloqué immédiatement mais son travail ne vaut pas 100% de la note maximale
  - soit donner un indice à l'élève -> dans ce cas l'élève n'est pas débloqué immédiatement (quelques secondes sont nécessaires), mais il ne perd pas de points
- **(B) Exclamation de l'élève** :
  - soit débloquer l'élève en lui donnant la solution -> dans ce cas l'élève perd des points, plus que dans le cas A correspondant (car l'élève n'a pas fait d'efforts)
  - soit lui donner un indice -> l'élève n'est pas débloqué immédiatement (quelques secondes sont nécessaires), et il perd quelques points, plus que dans le cas A correspondant, mais moins que si on lui donne la réponse
  - soit dire à l'élève que c'est un feignant et qu'il lise la doc et se débrouille -> dans ce cas le déblocage est possible mais pas systématique ; cette possibilité ainsi que le temps auquel se produit l'éventuel déblocage sont obtenus par tirage aléatoire
- **(C) Bavardage** :
  - soit dire à l'élève de se taire et de se remettre au travail -> dans ce cas l'élève perd des points ; de plus le déblocage est possible mais pas systématique ; cette possibilité ainsi que le temps auquel se produit l'éventuel déblocage sont obtenus par tirage aléatoire
  - soit dire à l'élève de sortir de classe -> l'élève ne gagne plus de points. Le score de jeu sera affecté d'abord parce que l'élève n'a pas une bonne note, d'autre part parce que l'enseignant n'a pas réussi a garder cet élève en cours ; il est pénalisé de la différence entre la note maximale (ex: 100%) et la note atteinte par l'élève au moment de son exclusion.

Lorsque le prof Krab tarde à s'occuper des élèves, les élèves s'énervent. Le taux d'énervement augmente donc avec le temps à partir du moment où l'élève est bloqué. Ce taux rediminue progressivement à mesure que l'élève travaille. Dans une version évoluée du jeu, vous pouvez également faire en sorte que les élèves voisins d'un élève qui bavarde et/ou qui s'énerve, s'énervent aussi.
L'énervement d'un élève (même lorsqu'il travaille et que son niveau d'énervement n'est pas redescendu à 0) augmente la probabilité qu'il bavarde. De plus, lorsqu'un élève est bloqué, s'il atteint son niveau d'énervement maximal, l'élève réalise une action inconsidérée :

- il sort de cours (il s'exclut lui-même) -> l'élève ne gagne plus de points. Le score de jeu sera affecté d'abord parce que l'élève n'a pas une bonne note, d'autre part parce que l'enseignant n'a pas réussi a garder cet élève en cours ; il est pénalisé de la différence entre la note maximale (ex: 100%) et la note atteinte par l'élève au moment de son exclusion.

- il provoque le bavardage de tous ses voisins

- ...

Un niveau de jeu est fini quand tous les élèves ont terminé leur travail ou sont sortis ou que le temps imparti pour un TP est terminé (vous pouvez par exemple fixer à 150 secondes le temps maximal, soit 1.5 fois le temps d'avancement normal de chaque élève (100 secondes)).

Les niveaux de jeu augmentent le nombre d'élèves, complexifient la disposition des bureaux dans les salles, rajoutent des obstacles, rajoutent des salles (impliquant que vous pouvez sélectionner la salle), rajoutent des professeurs, font réaliser un travail aux profs en parallèle (le prof doit finir son travail aussi avant la fin), transforment les élèves énervés en zombies agressifs qui poursuivent le prof, un dinosaure entre dans la salle, les PCs de la fac tombent en panne aléatoirement impliquant de nouvelles actions (déplacement de l'élève), le prof doit vraiment débugger du code qui s'affiche à l'écran pour débloquer la situation, ... Tout est possible, mais soyons clairs, le fonctionnement de la version "de base" sera déjà très bien !

b) Difficulté : ** à ***

Le jeu contient plus de composants (les bulles de dialogue émises par et appartenant aux élèves, les bureaux qui représentent l'avancement du travail des élèves, ...) et d'actions dont certaines sont ordonnées (suite d'actions de réponses à mener auprès des élèves ...). Les événements se produisant doivent être surveillés par les profs, mais aussi par les élèves voisins ...

2.3 Un shooter horizontal / un astéroids

a) Pitch Vous êtes dans un vaisseai spatial qui peut tirer des missiles ou tout autre projectile. Dans le cas d'un jeu de type Asteroids, tout se passe sur un seul écran de jeu. Le vaisseau peut se déplacer dans toutes les directions dans l'écran. De partout arrivent des objets, en particulier des astéroides, qu'il faut détruire.
Dans le cas d'un shooter (par exemple un shooter dit 'horizontal'), le vaisseau est aligné à l'horizontale (dirigé de gauche à droite) et semble se déplacer vers la droite. En réalité, c'est tout le décors qui se déplace vers la gauche. Des enemis apparaissent depuis la droite, soit immobiles (des éléments agressifs du décors, par exemple des tourelles canon), soit mobiles comme d'autres vaisseaux oud es créatures. Le but est d'arriver au bout du niveau évidemment.

Vous remarquerez que ce concept de jeu peut se transposer à d'autres situations, par exemple un jeu dans lequel le vaisseau est un globule blanc dans votre sang. Les énemis sont les virus et autres bactéries et parasites qui infectent votre hôte. Vous les détruises soit en les phagocytant (vous chercherez la définition en ligne), soit en envoyant dessus des immunoglobulines (là aussi, cherchez la définition). Vous pouvez vous faire aider par d'autres cellules immunitaires en communiquant avec elles grâce à des messagers chimiques, les cytokines. De plus, de nombreux éléments normaux mais gênants vous arrivent dessus comme des globules rouges, éléments qu'on peut détruire mais qu'il vaut mieux éviter de détruire pour ne pas affecter la survie de votre hôte.

b) Difficulté : asteroids : * / shooter horizontal : ** à ***

Dans le cas d'un asteroids, le jeu est assez simple a programmer car tout se passe sur un seul écran de jeu. Les difficultés viennent des mouvements d'une part (pas case à case), et des actions médiées par agents (le vaisseau du joueur emet des projectiles qui agissent sur les objets à détruire / modifier).

Dans le cas d'un shooter, la difficulté rajoutée est le changement continu d'écrans (scrolling) et la carte de grande taille ou la génération procédurale de carte.

3 Réalisation

3.1 Définitions préalables

Au cours de la conception du jeu, vous allez rencontrer des mots consacrés. J'en donne la définition ici :

- **Sprite** : c'est une marque de soda, mais c'est aussi un élément graphique que l'on peut éventuellement déplacer à l'écran. Un sprite peut être animé (ex: créature) ou non (ex: texte affiché avec une fonte graphique). C'est sa gestion en tant qu'objet graphique affichage qui le désigne comme sprite.
- **Décor** : à la différence du sprite le décor est souvent constitué d'une image de fond souvent immobile. Il est possible de rendre dynamique le décor avec l'usage de sprites (qui deviennent des éléments de décors mobiles ou animés).
- **Périphérique** : les périphériques désignent les éléments de l'ordinateur reliés à l'unité centrale. Il y a les périphériques d'entrée et les périphériques de sortie. Dans le cadre du jeu vidéo, nous aurons affaire aux périphériques d'entrées, à savoir le clavier et la souris, dont nous capturerons les états (touches pressées, relachées, maintenues, mouvements ...). Nous aurons aussi affaire au périphérique de sortie graphique (et éventuellement sonore).
- **Scrolling** : le scrolling est une opération graphique qui consiste à faire se déplacer le décor de façon fluide dans l'écran. Du nouveau décor apparaît d'un côté de l'écran vers lequel le sujet principal du jeu (le joueur) se déplace, tandis qu'à l'endroit opposé le décor disparaît.
- **Effet** : un effet graphique est une opération permettant de réaliser une transition d'une image vers une autre image, par exemple le fondu entre 2 images, le clignotement d'une image, le rétrecissement ou l'agrandissement d'une image ...
- **Gameplay** : le gameplay, c'est le coeur du jeu, à savoir ce qui fait du programme un jeu. Il intègre bien entendu les relations entre les composants du jeu (les créatures, le décor, ...), les contrôles que le joueur humain a sur la partie, les règles de jeu. Le soin qu'on y apporte contribue, avec le level design, au succès de l'expérience vidéoludique.
- **Level design** : le level design confère au jeu son intérêt ludique. Il s'agit de l'architecture des niveaux, de la façon dont le jeu va affecter la progression du joueur.

3.2 Composants essentiels du jeu et leur fonctionnement

Dans les 3 concepts présentés ci-dessus, il y a des points communs. Ils comportent tous au minimum de :

- un plateau de jeu intégrant éventuellement des décors passifs et positionnant les éléments actifs de jeu
- un joueur (élément actif) qui est habituellement représenté par un avatar (un personnage dans le jeu). Il peut ne pas y avoir d'avatar.
- un ensemble d'éléments actifs à vie propre (agents) qui peuvent influencer les états du joueur et qui peuvent s'influencer entre eux.
- des évenements soit scriptés, soit aléatoires
- des interactions directes (le prof clique sur la bulle de bavardage de l'élève pour lui dire de se taire) ou indirectes (le joueur envoie un objet sur un autre objet, par exemple un missile sur un astéroïde), les deux déclenchées par des actions clavier et/ou souris.

3.3 Ressources

Vous trouverez ici un certain nombre de ressources utiles pour la conception de votre jeu.

- Tutoriel **Ecrire un Game concept**
- La documentation de **Monogame**

- **Pixilart**, une app en ligne pour éditer des sprites
- **Piskel**, une autre app en ligne (et hors ligne) pour éditer des sprites
- **Universal LPC Spritesheet Generator**, une app en ligne pour générer des tuiles de sprites de personnages
- **Open Game Art**, un site vous donnant accès à de très nombreux objets graphiques (sprites, décors, ...)

4 Un premier projet Monogame - Jeu “MyGame”

La documentation de Monogame se trouve à ce lien.

4.1 Installation de Monogame

Nous allons installer la bibliothèque du framework Monogame. Pour ça, ouvrez un terminal et tapez la commande :
`dotnet new install MonoGame.Templates.CSharp`
Cela va installer la lib et les modèles de projets.

4.2 Création d’un projet Monogame (cross-platform)

Ce que nous voulons créer, c’est un projet MonoGame multi-plateformes, à savoir dont les graphismes seront gérés par la bibliothèque OpenGL.

Vous avez à votre disposition 2 façons de créer un projet dotnet, et en particulier ici un projet Monogame.

- **Depuis un terminal :** Dans un terminal, tapez la commande
`dotnet new mgdesktopgl -o NomDuProjet` où `NomDuProjet` est le nom que vous donnez à votre projet, par exemple `BasicMonoGame`.
- **Depuis Jetbrains Rider [Recommandé] :** Depuis le menu, faites :`New Solution -> Monogame Cross-Platform Desktop Application`. Cela ouvrira une fenêtre de dialogue dans laquelle vous indiquez le nom de votre Solution / Projet (mettez le même nom, par exemple `BasicMonoGame`), le chemin dans lequel vous créez la solution. Cochez les cases “Put solution and project in the same directory” et “Create Git repository”.

NB : Pour connaitre tous les types de projets pouvant être créés depuis dotnet (étant donné les libs installées) vous pouvez taper dans un terminal la commande
`dotnet new list`

Le projet créé contient une classe `Program` et une classe `Game1` qui hérite de la classe `Game` de la lib `Monogame`. Vous pouvez renommer cette classe (refactoring) bien entendu, par exemple `Game1` devient `MyGame`. Dans la suite des descriptifs ci-dessous, nous utiliserons ce nom là pour la désigner (`MyGame`).

Une classe (comme `MyGame`) héritant de `Game` permet de gérer un jeu. Elle contient (obligatoirement):

- Un attribut statique `Content`. Cet attribut de classe se réfère à un gestionnaire de contenu de type `ContentManager`. C’est à cette instance statique `Content` que vous transmettrez vos contenus graphiques (sprites, fontes ...).
- Les attributs dynamiques :
  - `_graphics : GraphicsDeviceManager` qui permet de lier le jeu (classe `Game` et dérivées) au gestionnaire de périphériques graphiques.
  - `_spriteBatch : SpriteBatch` qui permet de gérer les contenus graphiques (les sprites) compilés par mgcb (voir plus bas) : il met en interaction le périphérique graphique et les contenus (`Content`). Les dessins sont réalisés dans le spritebatch

- un constructeur qui instancie notamment un nouveau `GraphicsDeviceManager`
- 4 méthodes protégées surchargées (overrided) :
  - `void Initialize()` qui permet d'initialiser toutes les valeurs nécessaires à votre jeu. Elle est appelée au lancement du jeu.
  - `void LoadContent()` qui permet de charger les contenus dans l'instance `Content`. Elle est appelée au lancement du jeu.
  - `void Update(GameTime gameTime)` est une callback qui permet d'actualiser les évènements d'entrée (clavier, souris).
  - `void Draw(GameTime gameTime)` est une callback qui permet de dessiner.

4.3 Test du projet Monogame

Attendez que le projet s'ouvre complètement puis compilez (bouton Run). Si tout va bien, une fenêtre graphique bleue s'ouvrira. Pour sortir, pressez `Echap / Escape`.

4.4 Installation de l'outil mgcb (MonoGame Content Builder)

Une fois le projet Monogame créé, nous allons tester la présence de l'éditeur de ressources pour monogame et sinon l'installer.

- Ouvrez le terminal de Rider (petit icone `Terminal` en bas à gauche
- Dans le terminal tapez la commande `dotnet mgbc-editor`. Normalement un outil s'ouvre.
- Si ce n'est pas le cas, installez mgcb et mgcb-editor depuis le terminal de Jetbrains Rider en tapant `dotnet tool install -g dotnet-mgcb`.
  Testez à nouveau (point au dessus).

4.5 Ajout de ressources de jeu avec mgcb-editor

Nous allons maintenant ajouter des ressources qui seront prises en charges par MonoGame. Notez que dans votre jeu, toutes les ressources ne seront pas prises en charge par MonoGame, par exemple les fichiers XML... Ce que l'on désigne donc là par "ressources gérées par MonoGame", ce sont des images, sons, fontes, *etc* que les méthodes de la bibliothèque utilisent pour afficher des images (sprites) ou du texte et jouer des sons.

Pour être prises en compte, ces ressources doivent être intégrées dans le projet MonoGame et compilées par mgcb (le MonoGame Content Builder). Cet outil génère des fichiers à l'extension `xna` que l'on appelle contenu (`Content`). L'outil `mgcb` gère un projet nommé par défaut `Content.mgcb`. Ce projet peut être ouvert par `dotnet mgcb-editor Content.mgcb` ; il contient les ressources associées et permet de les compiler en contenus. Pour tout savoir sur les contenus MonoGame, référez vous à ce lien.

Dans le terminal de Jetbrains Rider, executez la commande `dotnet mgcb-editor &`. Cela ouvrira un editeur. De là, vous pouvez rajouter des items existants puis sauvegarder et compiler votre projet mgcb.

4.6 Ajout d'un sprite controlé par les périphériques d'entrée

Avec mgcb, ajoutez un sprite (montrant un personnage, un vaisseau spatial ou un objet quelconque pour l'instant), ce sprite vous servant à matérialiser le joueur (donc son avatar). Dans cet exemple, nous désignerons l'image du sprite par un nom générique (disons `sprite.png`) mais vous pouvez utiliser bien entendu un nom différent.

Créez une classe `Sprite`. Cette classe sera utilisée pour afficher le sprite. Nous ferons en sorte qu'un sprite soit instancié et mis en oeuvre par le jeu (classe `MyGame`). Voici la classe `Sprite` (fournie dans le TP) :

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;

namespace BasicMonoGame;

public class Sprite {

    private Texture2D _texture;
    protected Vector2 _position;
    private int _size = 100;
    private static readonly int _sizeMin = 10;
    private Color _color = Color.White;

    public Texture2D _Texture { get => _texture; init => _texture = value; }
    public int _Size { get => _size; set => _size = value>=_sizeMin?value:
        _sizeMin; }
    public Rectangle _Rect { get => new Rectangle((int) _position.X, (int)
        _position.Y, _size, _size); }

    public Sprite(Texture2D texture, Vector2 position, int size) {
        _Texture = texture;
        _position = position;
        _Size = size;
    }

    public void Update(GameTime gameTime){
        if (Keyboard.GetState().IsKeyDown(Keys.Up)) { /*...*/ }
        //...
    }

    public void Draw(SpriteBatch spriteBatch) {
        var origin = new Vector2(_texture.Width / 2f, _texture.Height / 2f);
        spriteBatch.Draw(   _texture, // Texture2D,
            _Rect, // Rectangle destinationRectangle,
            null, // Nullable<Rectangle> sourceRectangle,
            _color, //  Color,
            0.0f, //  float rotation,
            origin,  // Vector2 origin,
            SpriteEffects.None, // SpriteEffects effects,
            0f ); // float layerDepth
    }

}
```

Cette classe contient les attributs suivants :

- `_texture : Texture2D` : permet de charger une texture faisant partie des contenus compilés par mgcb
- `_position : Vector2` : permet de gérer la position de l’objet (coin supérieur gauche du sprite)
- `_size : int` : donne une taille au sprite
- `_sizeMin : int` : cet attribut statique permet de fixer une taille minimale qu’un sprite peut avoir
- `_color : Color` : permet de gérer la couleur du sprite. Par défaut cette couleur est blanche ; ainsi le sprite affiché est celui de l’image ; sinon l’image prend la couleur indiquée par cet attribut.

Vous remarquerez le rôle des propriétés `_Texture`, `_Size` et `_Rect` pour gérer l’accès de certains attributs ou rajouter un attribut calculé. La propriété `_Rect` permet de désigner un rectangle qui constitue la bounding box de l’objet.

La classe implémente les méthodes :

- `void Update(GameTime gameTime` : permet de gérer les modifications d'attributs comme la position, l'angle ou la taille en fonction de l'usage de touches du clavier ou de la souris.

- `void Draw(SpriteBatch spriteBatch` : permet d'afficher le sprite dans le spritebatch.

Pour mettre en oeuvre le sprite, vous devez l'instancier dans la classe `MyGame` et appeler ses méthodes `Update` (en transmettant le `gameTime` de `MyGame`) et `Draw` (en transmettant le `spriteBatch` de `MyGame`), comme suit :

```
1  using Microsoft.Xna.Framework;
2  using Microsoft.Xna.Framework.Graphics;
3  using Microsoft.Xna.Framework.Input;
4
5  namespace BasicMonoGame;
6
7  public class MyGame : Game {
8      private GraphicsDeviceManager _graphics;
9      private SpriteBatch _spriteBatch;
10     private Sprite _ship; // instance de Sprite
11
12     public MyGame() {
13         _graphics = new GraphicsDeviceManager(this);
14         Content.RootDirectory = "Content";
15         IsMouseVisible = true;
16     }
17
18     protected override void Initialize() {
19         base.Initialize();
20     }
21
22     protected override void LoadContent() {
23         _spriteBatch = new SpriteBatch(GraphicsDevice);
24         Texture2D shipTexture = Content.Load<Texture2D>("ship2");
25         _ship = new Sprite(shipTexture, new Vector2(150, 150));
26     }
27
28     protected override void Update(GameTime gameTime) {
29         if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.
               Pressed ||
30             Keyboard.GetState().IsKeyDown(Keys.Escape))
31             Exit();
32         _ship.Update(gameTime);
33         base.Update(gameTime);
34     }
35
36     protected override void Draw(GameTime gameTime) {
37         GraphicsDevice.Clear(Color.CornflowerBlue);
38         _spriteBatch.Begin(samplerState: SamplerState.PointClamp);
39             _ship.Draw(_spriteBatch);
40         _spriteBatch.End();
41         base.Draw(gameTime);
42     }
43 }
```

Modifiez la méthode `Update` de manière à faire bouger en avant, en arrière et sur les côtés votre vaisseau lorsqu'une touche du clavier est pressée.

Modifiez la méthode `Update` de manière à réduire la taille du vaisseau ou au contraire à la faire grossir. A cet effet, profitez en pour ajouter un attribut statique `_sizeMax` (de valeur 100 par exemple) et modifiez la propriété `_Size` de façon à ce que la taille `_size` ne puisse pas dépasser cette taille maximale.

4.7 Un mouvement plus réaliste

Ajoutez un attribut privé `_speed : double` qui donnera une vitesse à notre sprite. Nous allons nous en servir pour réaliser un mouvement réaliste ayant les caractéristiques suivantes :

- l'appui de la flèche du haut augmente la vitesse de la valeur 1. Cela fait avancer le vaisseau dans la direction vers laquelle il pointe
- l'appui de la flèche du bas augmente la vitesse (arrière) de la valeur -1. Cela fait reculer le vaisseau dans la direction opposée.
- l'appui de la flèche droite augmente la vitesse de la valeur 0.5. Cela fait bouger le vaisseau latéralement à droite (strafe). Le mouvement est moins rapide et doit diminuer plus vite.
- l'appui de la flèche gauche augmente la vitesse de la valeur 0.5. Cela fait bouger le vaisseau latéralement à gauche (strafe). Le mouvement est moins rapide et doit diminuer plus vite.
- la vitesse revient à la valeur 0 alors que le temps de jeu passe. Pour ça, si la vitesse est positive, on fait diminuer celle-ci de la valeur 0.05 à chaque update. Réciproquement, si elle est négative, on la fait augmenter de 0.05.

Modifiez la fonction update de votre Sprite comme suit :

```
1   public void Update(GameTime gameTime) {
2
3       if (Keyboard.GetState().IsKeyDown(Keys.Up)) {
4           _speed.X +=1.1f;
5       }
6       if (Keyboard.GetState().IsKeyDown(Keys.Down)) {
7           _speed.X -=1.1f;
8       }
9       if (Keyboard.GetState().IsKeyDown(Keys.Right)) {
10          _speed.Y +=0.5f;
11      }
12      if (Keyboard.GetState().IsKeyDown(Keys.Left)) {
13          _speed.Y -=0.5f;
14      }
15
16      _position.X = _position.X + _speed.X;
17      _position.Y = _position.Y + _speed.Y;
18      if (_speed.X > 0) _speed.X -= 0.05f;
19      if (_speed.X < 0) _speed.X += 0.1f;
20      if (_speed.Y > 0) _speed.Y -= 0.1f;
21      if (_speed.Y < 0) _speed.Y += 0.1f;
22  }
```

Une amélioration consiste à stocker la décélération (décroissance de la vitesse) dans une variable `_speedDec`. De même pour l'accélération avec une variable `_speedAcc`. Cela permet de régler les incréments décréments de vitesse avec des valeurs qu'on peut modifier, contrairement aux valeurs fixes proposées dans le listing ci-dessus. Ces variables doivent être des propriétés ou être des attributs privés contrôlés par des propriétés de façon à imposer des limites minimale et maximale à l'accélération et à la décélération, par exemple des valeurs comprises entre $0.0f$ et $1.0f$. Elles devront bien entendu être instanciées et initialisées dans le constructeur.

4.8 Modification : création d'une classe GameObject et d'une classe Player qui héritent du Sprite

Nous allons maintenant utiliser cette base pour réaliser un principe de jeu plus ambitieux.

Vous allez :

- Créer une classe `GameObject` qui hérite de la classe `Sprite`. Dans ce contexte, un `Sprite` n'est plus qu'un objet affichage mais ne dispose plus de controles, de vitesse, de caractéristiques autre que celles de son affichage. Un `GameObject` est un objet de jeu pouvant se déplacer ; c'est lui qui dispose d'une vitesse, d'une accélération, d'une décélération. La classe `Sprite` ne doit donc plus

réaliser d'update de mouvement ; c'est le `GameObject` qui met à jour la position en fonction de la vitesse.

- Créer une classe `Player` qui hérite de la classe `GameObject`. Seul le le `Player` est doté du contrôle des mouvements. C'est aussi sa fonction d'update qui fait décélérer la vitesse. Cette fois, c'est à ce joueur qu'est associées la texture de vaisseau `ship2` qui est fournie.
- A ce stade votre vaisseau est contrôlable mais il peut traverser les bords de l'écran (et être perdu de vue). A vous de faire le nécessaire pour ne pas perdre votre vaisseau. Il y a plein de solutions, par exemple :
  - faire chuter la vitesse à 0 (empêcher le déplacement) lorsqu'on arrive aux abords de l'écran et qu'on s'y dirige (méthode la plus simple)
  - recentrer l'ensemble des objets, dont le vaisseau, avec un mouvement inverse lorsque le vaisseau s'arrête
  - idem, mais l'action de recentrage est déclenchée par l'appui d'une touche
  - afficher un sprite de flèche indiquant la direction où se trouve le vaisseau en dehors de l'écran
  - afficher une minimap qui localise le vaisseau hors de l'écran mais dans le jeu (donc le jeu à d'autres limites que l'écran)
- Ajouter une classe `Creature`. Cette classe se comporte de façon semblable sauf que les créatures ont leur mouvement propre. A vous de voir quel mouvement vous souhaitez implémenter. Vous pouvez utiliser la texture `Virus` fournie pour cette créature. Eventuellement, rajoutez un type énuméré qui décrit différentes créatures (ex: virus, parasite, bactérie ...) et donnez un type à la construction de votre créature. Selon son type, la créature aura des mouvements différents et une texture différente.
- Créez une classe `Projectile`. A vous de voir comment elle doit se comporter, de qui elle hérite ... Evidemment, l'idée est que l'appui d'une touche du clavier déclenche la production d'un projectile (ex: missile ...) par le vaisseau, ce missile pouvant avoir (selon son type) divers comportements comme se déplacer tout droit jusqu'à entrer en collision avec une créature, ou chercher les créatures, ou cibler une créature ... Cela implique que vous devrez gérer les collisions.

5 Réalisation du jeu principal

5.1 Conceptualisation

Réfléchissez au concept de jeu que vous souhaitez créer. Ecrivez sa fiche signalétique. Vous devrez produire une fiche signalétique complète comme montré dans le site Tutoriel **Ecrire un Game concept**. Crééé une première version complète que vous affinerez par la suite.

5.2 Modélisation UML

Ecrivez le diagramme de classes et le diagramme de séquence de votre jeu. Cette modélisation pourra se faire par incréments au fur et à mesure que votre réalisation évoluera. Néanmoins, vous devez, avant de commencer le code proprement dit, avoir modélisé dans les grandes lignes les composants de votre jeu et son fonctionnement.

5.3 Le reste ...

Le reste est à venir !

6 Rendus

Les rendus seront obligatoirement :

- La fiche signalétique de votre jeu au format PDF exclusivement (tout autre format ne sera pas accepté). La fiche signalétique deva inclure le nom des auteurs.

- Le projet C# incluant :
  - Le code source nettoyé
  - La documentation technique (documentation des classes et méthodes)
  - Les ressources graphiques, textuelles (dont fonts) et sonores
  - Les diagrammes UML de classe et de séquence du jeu
  - Une vidéo courte (moins de 2 minutes) de présentation de votre jeu
- le lien vers le dépot Github ouvert.

L'ensemble sera compressé dans **un seul fichier au format ZIP** (exclusivement ! pas de `.rar`, pas de `tar.gz` ...) dont le nom sera composé des noms (première lettre du prénom + nom) des auteurs, par exemple : `CFouard_ADemeure_ALagoutte_NGlade_.zip`

7 Conclusion

Vous êtes fier-e de votre jeu et de ce que vous avez entrepris pour y arriver ?!

C'est bien et ça peut vous donner envie d'aller plus loin comme *Lou Lubie* avec le projet *Rose in the woods* et sa *Petite Fabrique du Jeu Vidéo*.

En attendant, pensez à utiliser ce que vous aurez appris (méthodologie, organisation, formalisation, technos ...) dans votre projet intégrateur.

Chapitre V

Projet Trajectoires

1 Présentation générale du projet

Le projet consiste à modéliser en XML, puis en C#, un ensemble de particules bougeant de manière aléatoire dans un environnement de simulation, et à enregistrer leurs trajectoires. La simulation sera initiée grace à un nombre de particules et une définition des caractéristiques de l'environnement. Elle génèrera les particules. Lors de son arrêt elle enregistrera d'une part les trajectoires individuelles des particules dans `Trajectories.xml`, mais également les positions des particules dans `Particles.xml`. La simumlation de particules pourra éventuellement être affichée de façon dynamique sous la forme d'une application MonoGame. De plus, les enregistrements des trajectoires feront l'objet d'une transformation XSLT vers une image SVG qui permettra leur affichage dans un navigateur (HTML).

Ce travail vous donnera l'occasion d'intégrer vos connaissances en XML, XML Schema, XSLT avec l'usage de C#, de ses parsers et de la sérialisation.

2 Réalisation

2.1 Modélisation XML Schema

Vous allez devoir modéliser plusieurs types dont la plupart des instances XML vous sont fournies. Les schémas XML devront valider ces instances sans modification.
Il est recommandé de passer par une étape de modélisation UMC (plant UML).

Les types à modéliser sont :

- Coordinate
- BoundingBox
- Environnement
- Particle
- Particles
- Trajectory
- Trajectories
- Simulation

Notez qu'un `Environnement` est une `BoundingBox`, au même titre qu'une `Particle`. Bien entendu, `Particles` contient une liste de `Particle`, `Trajectories` contient une liste de `Trajectory`, une `Simulation` contient un `Environnement`

2.2 Feuilles de transformation XSLT

Ecrire une feuille de transformation qui génère une figure au format SVG (Documentation SVG) comme suit. La figure enregistrée dans un fichier `trajectoires.svg` contient :

- un rectangle coloré qui représente l'environnement
- un ensemble de lignes (`polyline`) qui représentent chacune une trajectoire.

Il est recommandé d'écrire un SVG test à la main et de l'afficher en HTML, comme cela a été fait en cours.

Ecrire une deuxième feuille qui génère directement une page HTMl qui affiche cette figure.

2.3 C# - Validation de schéma et feuilles XSLT

Ecrivez une classe C# nommée `Simulation` qui valide les schemas de la simulation, des particules et des trajectoires.

Ecrivez dans cette même classe une méthode qui applique sur les données de trajectoires la feuille de transformation XSLT générant du SVG.

Idem avec une méthode générant le HTML correspondant.

2.4 C# - Sérialisation

Ecrivez les classes suivantes de telle manière à ce qu'elles soient capable de sérialiser les types XML correspondants.

- BoundingBox
- Environnement
- Particle
- Particles
- Trajectory
- Trajectories
- Simulation

Le type `Coordinate` n'a pas besoin d'être codé en C# car il est substitué par le type existant `Vector2`.

Partie C

Travaux pratiques

Chapitre VI

TP - Premier contact avec UML, XML et XML Schema

1 Démarrage

Au début de ce module, il est possible que certains ne disposent pas encore d'une adresse universitaire (etu.univ-grenoble-alpes). Dans ce cas, vous pourrez réaliser les premiers TPs sous **Netbeans** qui est disponible sur les machines de l'université.

Sinon, vous utiliserez **Jetbrains** (idéalement **Rider**, sinon **IntelliJ**). Ce logiciel est payant normalement, mais gratuit pour qui dispose d'une adresse universitaire.
JetBrains est installé sur les machines de la fac. Vous pouvez aussi décider de l'installer sur votre machine personnelle (dans ce cas, installez **Jetbrains Rider**).
Dans les deux cas, pour pouvoir l'utiliser, vous devez disposer d'une licence.
Si vous disposez d'une adresse universitaire grenobloise valide, vous devez vous rendre à cette adresse Jetbrain license et cliquer sur **Apply now** dans le bandeau violet titré **Get free access to all developer tools from JetBrains!**. Suivez les instructions et récupérez votre numéro de licence. Vous pouvez ensuite l'utiliser pour l'usage de Jetbrains.

2 Création d'un projet JetBrains Rider / Netbeans

Si vous pouvez utiliser Jetbrains, vous devez un projet C#. Sinon vous pouvez pour commencer, générer un projet Java sous Netbeans. Ce projet Java sera ensuite rapidement remplacé pr un projet C# sous Jetbrains Rider.

2.1 Jetbrains Rider

Creez un nouveau projet de type `dotnet console` : depuis le menu : `New solution`. Incluez la gestion de git (création de repository git) ; cela vous permettra de sauvegarder l'avancée de votre projet. Cf la figure VI .1.

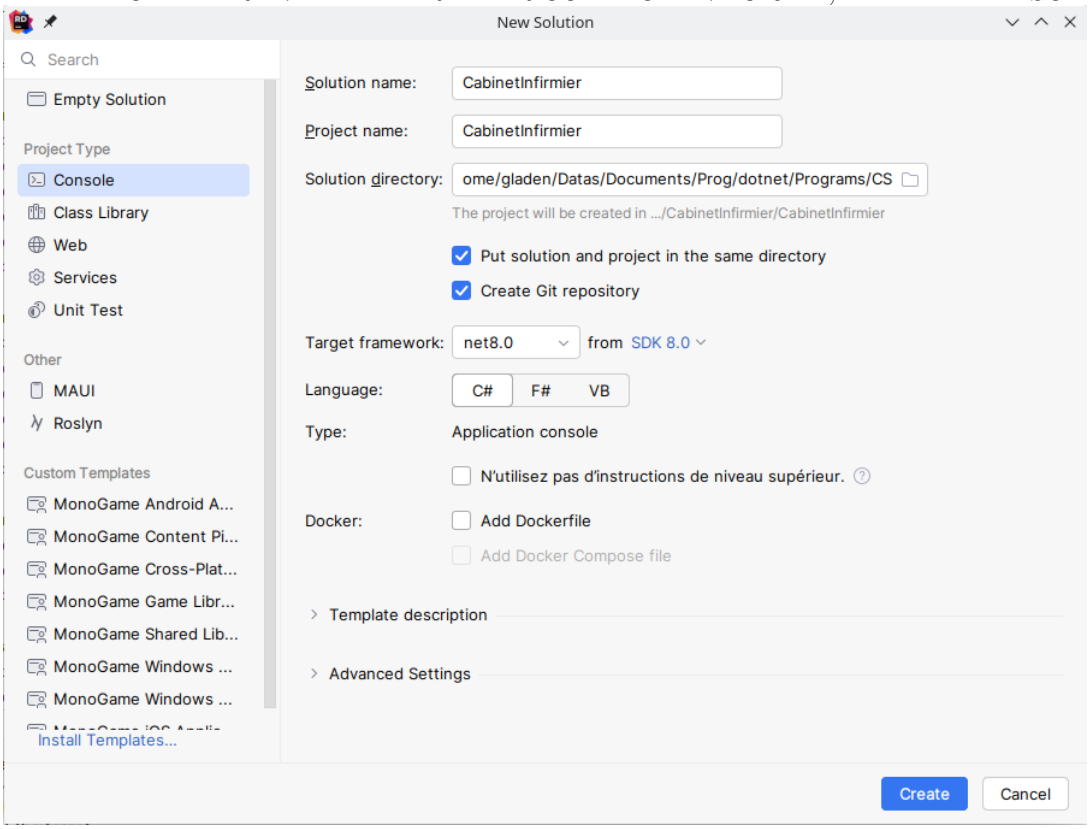

Figure VI .1: **Création du projet (solution) Cabinet Infirmier sous Jetbrains Rider)**.

Dans ce projet, dans le dossier `src`, vous créerez l'arborescence montrée figure VI .2.

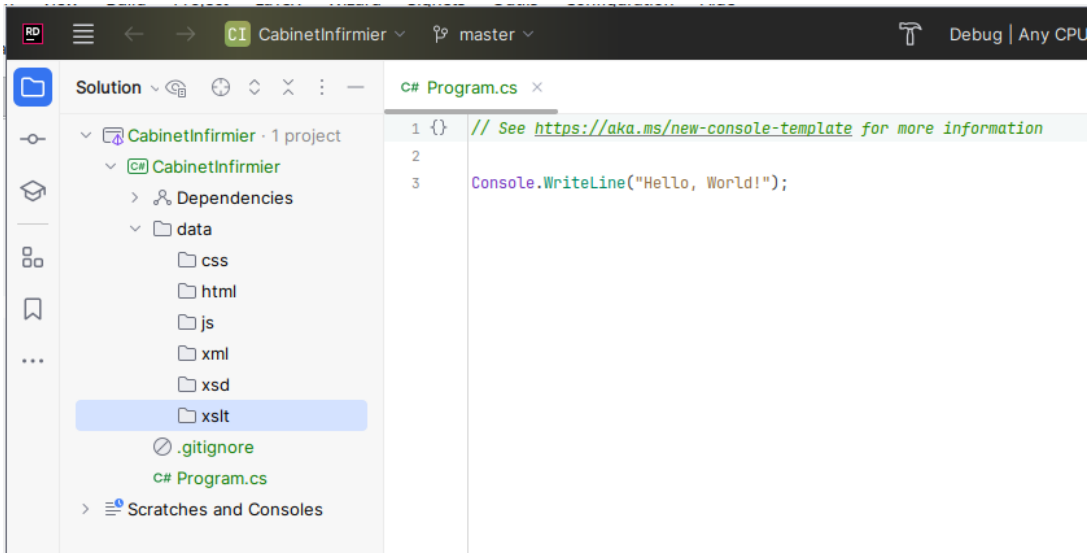

Figure VI .2: **Arborescence des données dans le projet Cabinet Infirmier**.

2.2 Netbeans (solution de secours en début d'année)

Pour chaque Exercice, créez un projet Java de type `Java with Ant`. Nommez le `Exo1`, `Exo2` ...

Pour le projet `Cabinet Infirmier`, créez un projet Java de type `Java with Ant` que vous nommerez `CabinetInfirmier`. Dans ce projet, dans le dossier `src`, vous créerez l'arborescence montrée figure VI .3.

NB : dès que vous pourrez passer sous Jetbrains, vous créerezun projet C# dans lequel vous copierez l'ensemble de cette arborescence.

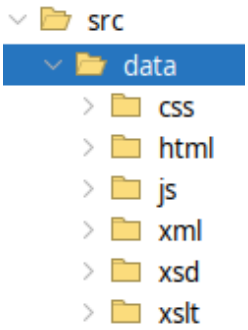

Figure VI .3: **Arborescence des données dans le projet Cabinet Infirmier**.

3 Exercices de prise en main

Ces exercices ne sont pas rendus et ne font pas l'objet de notation. Ils sont cependant votre assurance de comprendre le cours en vous exerçant. Il est indispensable de les faire avant de s'attaquer à un quelconque projet.

3.1 Un premier document XML ... et son diagramme UML (Exo1)

3.1.1 1ère étape.

Créez un nouveau document XML nommé `planning.xml`. Testez l'usage des outils de l'IDE (voir chapitre II) pour :

- vérifier la conformité XML
- vérifier la validité par rapport à un schéma (votre document XML de base n'étant pas contraint par un schéma, il ne se passera pas grand chose d'intéressant !)

3.1.2 2ème étape.

Considérons le document suivant :

```
<?xml version="1.0" encoding="UTF-8"?>
<week>
  <date>20 oct</date>
  <tutorial>Algorithmique : statistiques</tutorial>
  <lecture>Boucles et conditions en Java
  </Lecture>
</week>

<holidays>
  <date>27 oct</date>
</holidays>

<week>
  <date>3 nov</date>
  <tutorial>Algorithmique : tableaux</tutorial>
  <Lecture>Les tableaux N dimensions
  </lecture>
</week>
```

Remplacez le contenu du document `planning.xml` par celui-ci. Le document est il bien formé ? Pourquoi ? Prenez les mesures qu'il faut pour corriger le document si nécessaire.

3.1.3 3ème étape.

Réalisez un diagramme UML de ce document, sur papier d'abord, sur PlantUML ensuite.

a) Papier. Réalisez scrupuleusement toutes ces étapes :

- Commencez par identifier à partir du document le ou les types dont vous avez besoin : listez les.
- Identifiez quels types seront des restrictions (des types simples restreints) et comment.
- Dessinez sur papier un schéma UML contenant tous les types et leurs relations (composition, ...)
- N'oubliez pas d'englober ces types dans un namespace. Le nom du vocabulaire sera `http://www.univ-grenoble-alpes.fr/l3miage/planning`.

b) UML - PlantUML Vous allez maintenant implémenter votre diagramme dans le langage de script `PlantUML`. Vous trouverez toute la documentation nécessaire sur ce site, en particulier ici : documentation pour les diagrammes de classe.

Pour tester vos scripts et générer vos diagrammes sous forme d'images, vous devez les soummettre dans le service en ligne PlantUML.

Progressez doucement, par petites implémentations.

3.2 Un premier Schema XML, son diagramme UML ... et son instance XML (Exo 2)

Cette fois, vous partez d'un schéma XML. A partir de ce schéma que vous corrigerez si nécessaire, vous concevrez un diagramme UML (papier d'abord, PlantUML ensuite), puis génèrerez une instance de document XML que vous complèterez (données valides par rapport au schéma) ; comme vous avez des profs sous la main, trouver leurs noms, prénoms et spécialités pour la filière Miage ne devrait pas poser de problèmes.

Voici ce Schema XML :

```
1  <?xml version="1.0"?>
2  <xs:schema xmlns:ens="http://www.univ-grenoble-alpes.fr/enseignants"
3          xmlns="http://www.w3.org/2001/XMLSchema"
4          targetNamespace="http://www.w3.org/2001/XMLSchema"
5          elementFormDefault="qualified">
6
7      <!-- Element racine -->
8      <element name="enseignants" type="Enseignants"/>
9
10     <!-- Le type Enseignants contient une séquence d'"Enseignant"
11         ainsi que la filière et l'année dans laquelle ils enseignent.
12     -->
13     <complexType name="Enseignants">
14         <attribute name="filiere" type="string"/>
15         <attribute name="annee" type="gYear"/>
16         <sequence>
17             <element name="enseignant" type="ens:Enseignant" maxOccurs="
                   unbounded"/>
18         </sequence>
19     </complexType>
20
21     <!-- Le type Enseignant contient un nom, un prénom et une spécialité
22         Il est identifié par une référence.
23      -->
24     <complexType name="Enseignant">
25         <sequence>
26             <element name="nom" type="ens:Denomination"/>
```

```
            <element name="prenom" type="ens:Denomination"/>
            <element name="specialite" type="ens:Specialite"/>
        </sequence>
        <attribute name="ref" type="xs:string"/>
    </complexType>

    <!-- Le type Denomination contraint les noms et prénoms -->
    <simpleType name="Dénomination">
        <restriction base="string">
            <pattern="[A-Z][a-z]*"/>
        </restriction>
    </simpleType>

    <!-- Le type Specialite contraint le choix des spécialités -->
    <simpleType name="Specialite">
        <restriction base="string">
            <enumeration type="APO"/>
            <enumeration type="FDD-XML"/>
            <enumeration type="IHM"/>
            <enumeration type="RO"/>
        </restriction>
    </simpleType>

</xs:schema>
```

Question subsidiaire : quelles sont les pistes d'amélioration pour un tel schéma ? Comment procèderiez vous ?

3.3 Un premier UML ... et tout le reste (Exo 3)

Cette fois, nous allons partir d'un énoncé. Ce sera à vous d'imaginer un diagramme UML (papier + PlantUML) permettant de modéliser cette situation. A partir de ce que vous aurez proposé comme diagramme, créez le schéma XML complet correspondant en fixant un namespace, puis générez un fichier XML que vous renseignerez.

Voici l'énoncé :
On veut concevoir un jeu vidéo se passant dans une salle de classe. Le jeu consiste pour un personnage, le professeur, à répondre aux questions d'étudiants en TP de FDD-XML atablés chacun devant un ordinateur. Chaque salle de classe représente un niveau de jeu différent. On souhaite ici modéliser un niveau de jeu.
Un niveau de jeu tel qu'on souhaite le modéliser contient :

- un temps de jeu limite de type entier
- la structure d'une salle de classe

Une salle de classe est constituée d'un nombre compris entre 2 et 20 rangées (ou lignes) de tables.
Une ligne de tables est modélisée par une chaine de caractère contenant un nombre indéterminé de valeurs `0`, `1` ou `2`.

4 Projet Cabinet Infirmier

- Lisez **très attentivement** la description du projet infirmier (chapitre III
- Durant les 4 séances à venir, vous réaliserez la partie modélisation (section 5 du chapitre III) du projet Cabinet Infirmier. Attention, ce travail (partie XML du cabinet infirmier) sera récupéré vers la moitié du module et noté. Vous devez donc gérer votre temps en évaluant combien de temps vous passez sur chaque réalisation.

Chapitre VII

TP - XSLT - XSD

Cette double séance sera fort chargée ! Vous commencerez à pratiquer XSLT, continuerez à mettre en oeuvre des schémas XML et les instances correspondantes. De plus, c'est le début du projet Jeu Vidéo.

1 Exercices de prise en main

Ces exercices ne sont pas rendus et ne font pas l'objet de notation. Ils sont cependant votre assurance de comprendre le cours en vous exerçant. Il est indispensable de les faire avant de s'attaquer à un quelconque projet.

Vous devez aller chercher dans la documentation les renseignements nécessaires. L'essentiel est dans votre cours ou en ligne sur W3Schools.

1.1 Première transformation XSLT (Exo1) : un rectangle

1.1.1 Modélisation.

Modélisez (UML + XSD) un rectangle dont voici une instance :

```
<?xml version="1.0" encoding="UTF-8"?>
<rectangleData
        xmlns="http://www.univ-grenoble-alpes.fr/rectangle"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.univ-grenoble-alpes.fr/rectangle
            Rectangle.xsd">
    <x>6</x>
    <y>40</y>
    <width>10</width>
    <height>20</height>
</rectangleData>
```

Vous prendrez soin de bien vérifier la validité de votre document, ainsi que sa conformité.

1.1.2 transformation HTML

Créez une feuille de transformation XSLT s'appliquant à l'instance `rectangle.xml`. Cette transformation doit produire une page HTML affichant les caractéristiques de votre rectangle (`Position (X,Y), Largeur L, Hauteur H`).

Vérifiez que la transformation fonctionne en l'appliquant sur le document xml.

Ajoutez une feuille de style CSS et appliquez à nouveau la transformation.

1.1.3 Calcul lors de la transformation.

Modifiez votre feuille XSLT de manière à ce qu'elle affiche en plus l'aire du rectangle. Le calcul doit être fait lors de la transformation.

1.2 Deuxième transformation XSLT (Exo2) : Météo

La mairie de Villard de Lans, petite ville du massif du Vercors et station de sports toutes saisons, souhaite mettre en place un flux d'informations `RSS` informant sur la météorologie, à raison de deux relevés par jour, un a 6h00 du matin, l'autre à midi. Ces flux doivent renseigner sur 3 points : (1) *la température* incluant mesure de température (une température est une valeur supérieure à -273°C) et éventuellement la valeur de l'isotherme zéro (altitude en m à laquelle il fait 0°C, cette mesure n'étant pas nécessairement fournie), (2) *les précipitations* incluant le niveau de précipitations en *mm* d'eau et le type de précipitations (*aucune*, *pluie*, *grésil*, *grêle*, *neige*), et (3) *l'anémométrie*, c'est à dire la vitesse du vent en *km/h*, avec éventuellement la vitesse des rafales de vent (cette mesure n'étant pas nécessairement fournie), et la direction du vent avec 8 directions possibles (*N*, *NO*, *O*, *SO*, *S*, *SE*, *E*, *NE*), cette information n'étant pas nécessairement fournie (par exemple lorsqu'il n'y a pas de vent). Les données sont stockées au format `XML`, selon un schéma suffisamment générique pour être par la suite étendu à la communauté de communes de ce massif montagneux. Un fichier par mois est produit et est nommé `meteo-ville-mmyyyy.xml` (par exemple `meteo-VillarddeLans-122017.xml`). En plus des bulletins météo, chaque fichier contient le renseignement sur la période de l'année (mois et année), ainsi que les informations sur le site sur lequel sont effectuées les mesures. Les informations géographiques sont au format `KML`. *NB*: Tous les renseignements nécessaires à la modélisation se trouvent dans ce texte !

1.2.1 Modélisation des données météo en `XML et XMLSchema`

Dans cette partie, nous allons modéliser les données météorologiques convenablement. On fourni ci-dessous une instance de fichier XML tel que souhaité pour le mois de décembre ; on ne montre ici que les bulletins du 18 décembre 2017.
→ Fichier ***meteo-VillarddeLans-122017.xml*** :

```
<?xml version="1.0" encoding="UTF-8"?>
<met:météo
    xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
    xmlns:kml='http://www.opengis.net/kml/2.2'
    xmlns:gx="http://www.google.com/kml/ext/2.2"
    xmlns:met='http://www.meteo-france.fr/bulletin'
    xsi:schemaLocation='http://www.meteo-france.fr/bulletin ./meteo.xsd'
    url="http://www.espace-villard-correncon.fr/meteo-villard-de-lans.htm">
    <met:période année="2017" mois="12"/>
    <met:lieu>
        <kml:name>Villard de Lans</kml:name>
        <kml:description>Alpes Vercors Nord</kml:description>
        <kml:Point>
            <gx:altitudeMode>relativeToSeaFloor</gx:altitudeMode>
            <kml:coordinates>5.550886412795313,45.06638495609347,1008</kml:coordinates>
        </kml:Point>
    </met:lieu>
    <!-- Bulletins d'exemple -->
    <met:bulletin date="2017-12-16T06:00:00+01:00">
        <met:température>-6.5</met:température>
        <met:anémométrie>
            <met:vitesse>10</met:vitesse>
        </met:anémométrie>
        <met:précipitations type="neige">
            <met:niveau>15.0</met:niveau>
        </met:précipitations>
    </met:bulletin>
    <met:bulletin date="2017-12-16T12:00:00+01:00">
        <met:température isothermeZero="600">-2.0</met:température>
        <met:anémométrie direction="NE">
            <met:vitesse>30</met:vitesse>
            <met:rafales>55</met:rafales>
        </met:anémométrie>
        <met:précipitations type="aucune"/>
    </met:bulletin>
    <!--
        A COMPLETER : Bulletins du 21/12/17
    -->
</met:météo>
```

On donne également le squelette du schéma XML correspondant:
→ Fichier ***meteo.xsd*** :

```
<?xml version="1.0"?>
<xs:schema version="1.0"
           xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:met="http://www.meteo-france.fr/bulletin"
           xmlns:kml="http://www.opengis.net/kml/2.2"
           targetNamespace="http://www.meteo-france.fr/bulletin"
           elementFormDefault="qualified">

    <xs:import namespace="http://www.opengis.net/kml/2.2"
               schemaLocation="http://schemas.opengis.net/kml/2.2.0/ogckml22.xsd"/>
    <xs:import namespace="http://www.google.com/kml/ext/2.2"
               schemaLocation="http://code.google.com/apis/kml/schema/kml22gx.xsd"/>

    <!-- élément racine -->
    <!-- Type Météo -->
    <!-- Type Période -->
    <!-- Type Bulletin -->
    <!-- Type ValeurTempérature -->
    <!-- Type Température -->
    <!-- Type Anémométrie -->
    <!-- Type Direction -->
    <!-- Type Précipitations -->
    <!-- Type TypePrécipitation -->
    <!-- Type DoublePositif -->

</xs:schema>
```

Modélisez les types (i) `TypePrécipitation`, (ii) `ValeurTempérature` et (iii) `Direction` qui représentent respectivement (i) les différentes types de précipitations, (ii) les valeurs que peuvent prendre la température et (iii) les différentes directions du vent. Ecrivez également un type `DoublePositif` qui sera utilisé pour les valeurs de vitesse du vent et les niveaux de précipitations (positives). Ecrivez enfin un type `Période` dont les attributs `année` et `mois` correspondent aux types XML Schéma standard `xs:gYear` et `xs:gMonth`.
Modélisez les types `Précipitations`, `Température` et `Anémométrie` (vitesse du vent) de telle manière que cela corresponde à l'instance de document et aux renseignements donnés dans le texte. Ecrivez également le type `Bulletin` et le type `Météo`. Ecrivez enfin l'élément racine.
NB: Le type XML Schéma standard de date employé, `xs:dateTime`, encode la date, l'heure et le fuseau horaire ; la date et l'heure sont séparés par la lettre 'T'. Le lieu est de type `kml:PlacemarkType` du langage `KML`.

1.2.2 Transformation `XSLT` vers des documents `RSS`

On souhaite transformer chaque jour les bulletins du jour au format `RSS` comme dans le document ci-dessous.
→ Fichier ***meteo.rss*** :

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0"
     xmlns:kml="http://www.opengis.net/kml/2.2"
     xmlns:met="http://www.meteo-france.fr/bulletin">
    <channel>
        <title>Météo</title>
        <date>2017-12-14</date>
        <link>Météo des neiges [Villard de Lans] : http://www.espace-villard-correncon.fr/meteo-
            villard-de-lans.htm</link>
        <description>Bulletins météo du 2017-12-14 pour Villard de Lans, secteur Alpes Vercors Nord</
            description>
        <item>
            <title>Bulletin météo du matin</title>
            <pubDate>2017-12-14 - 06:00:00 - GMT+01:00</pubDate>
            <description>
                Température : -6.5 degrés
                Précipitations [neige]: 15.0 mm
                Vent : 05 km/h
            </description>
        </item>
        <item>
            <title>Bulletin météo de l'après-midi</title>
            <pubDate>2017-12-14 - 12:00:00 - GMT+01:00</pubDate>
            <description>
                Température : 2.0 degrés
```

```
                Précipitations [aucune]
                Vent : 30 km/h avec rafales à 55 km/h
            </description>
        </item>
    </channel>
</rss>
```

Créez une feuille de transformation XSLT `meteo2RSS.xslt` de façon à obtenir un fichier `RSS` semblable au document d'exemple donné ci-dessus. La date sélectionnée est transmise par paramètre via l'application. La feuille de transformation doit permettre notamment l'affichage de 1 à 2 bulletins par jour, selon disponibilité. Elle ne doit évidemment comporter qu'un seul template pour les deux bulletins.
NB: vous pourriez avoir besoin de la fonction XPath `substring(s:string,start:int,offset:int)`.
NB: la structure de branchement if/else en XSLT s'obtient avec `<xsl:choose><xsl:when test="....">....</xsl:when`
Préalable : Qu'est ce que RSS ? <-> quel sera la méthode de l'élément output de votre feuille ?

2 Projet Cabinet Infirmier

- Lisez **très attentivement** la suite (partie XSLT) de la description du projet infirmier (chapitre III
- Durant les séances à venir, vous réaliserez la partie XSLT du chapitre III .

3 Projet Jeu vidéo

Lisez le projet *Jeu vidéo* attentivement.

Commencez la réalisation (conceptualisation).

Chapitre VIII

TP - C#, XSD et XSLT

Ce TP va couvrir plusieurs séances (3 séances environ). La part évaluée de chaque partie sera indiquée ci-dessous.
En bref, au cours de ces séances vous :

- réaliserez un petit exercice en C# continuant l'exercice sur le Rectangle. Cet exercice intègre l'écriture de nouvelles feuilles de transformation XSLT. [*Temps estimé : 1h30 maximum*]
- commencerez le mini projet **Trajectoires**. Ce mini projet inclus l'écriture de schémas XML, de feuilles de transformation XSLT, de code C#. Il préfigurera l'architecture des entités peuplant un jeu vidéo. Il vous permettra de vous exercer en programmation C#, sur les parsers, sur la sérialisation, sur la programmation style fonctionnel ... Ce projet couvrira 5 à 6 séances. [*Le temps estimé de la réalisation du mini projet relative à ce TP (modélisation XSD + feuilles XSLT seulement) est de : 4h*]
- commencerez la réalisation du projet Jeu vidéo incluant :
  - la création d'un premier projet C# avec Monogame (`MyGame`) et l'affichage et la fonctionnalisation d'un sprite. [*Temps estimé : 2h maximum*]
  - la rédaction de la fiche signalétique du jeu et la réalisation d'une première version des diagrammes UML de classe et fonctionnel. [*Temps estimé : 2h maximum*]
- finirez la partie XML du Projet cabinet [*Vous devriez être à jour sur toute la partie XML, XSD, XSLT. Le temps estimé pour la partie restante (valisation et transformation C#) : 30min - 1h maximum*] :
  - finir XSLT
  - ajout clefs unicité et existence
  - creer classe cabinet
  - ajout validation schema et xslt en C#

a) Organisation des séances (recommandation). Il est recommandé de réaliser le travail à faire de la façon suivante (dans l'ordre) :

- **1ère séance.**
  - Faire l'exercice **Rectangle** sans l'écriture des nouvelles feuilles de transformation (SVG).
  - Finir ce qui est demandé pour le projet `Cabinet Infirmier`
  - Faire l'exercice Jeu Vidéo `MyGame` sans la partie `Un mouvement plus réaliste`
  - Lire le mini-projet `Trajectoires`
  - Réfléchir à votre fiche signalétique de jeu et commencer (sur papier) à poser les bases de son organisation (diagrammes).

- **2ème séance.**
  - Finir l"exercice **Rectangle** (écriture des 2 nouvelles feuilles de transformation (SVG)).
  - Finir l'exercice Jeu Vidéo `MyGame` incluant la partie `Un mouvement plus réaliste`
  - Réaliser une première version complète de la fiche signalétique de votre jeu et commencer à réaliser les diagrammes UML de classe et de séquence.
  - Commencer à modéliser les types demandés dans le mini-projet `Trajectoires`
- **3ème séance.**
  - Finir la modélisation XML du projet `Trajectoires` et faire les feuilles de transformation. Commencer l'écriture des classes C# correspondantes.
  - Commencer la programmation de votre jeu

1 Exercice - Rectangle

Vous réaliserez un petit exercice en C# continuant l'exercice sur le Rectangle (que vous avez précédemment modélisé):

- créer un projet C# avec une classe `Rectangle` correspondant au schéma XML que vous avez précédemment réalisé.
- ajoutez à cette classe une méthode de validation du fichier `Rectangle.xsd`
- ajoutez une méthode d'application de la transformation XSLT (`Rectangle.xslt`)
- écrire une autre feuille de transformation de ce rectangle en rectangle au format SVG. La sortie doit être un fichier svg.
- idem, mais cette fois votre svg doit être inclus dans un fichier HTML

Les informations nécessaires se trouvent dans le cours sur les parsers (à la fin).

2 Mini-projet Trajectoires

Vous commencerez le mini projet **Trajectoires** :

- Les instances XML sont fournies
- Modélisez les types correspondants en XML Schema : Coordinate, BoundingBox, Particule, Trajectoires ...
- XSLT : faire un XSLT qui traduit les Trajectoires en SVG

3 Projet Cabinet Infirmier

Liste des choses à faire :

- finir les XSLT si ce n'est pas déjà fait.
- ajoutez les clefs d'unicité et d'existence pour garantir l'unicité des patients d'une part, des infirmiers d'autre part, et garantir l'existence des infirmiers dont l'identifiant est indiqué dans les visites des patients.
- creer une classe C# cabinet
- ajoutez à cette classe 2 méthodes, une pour valider son Schema XML, l'autre pour appliquer la transformation XSLT de ses données XML (page de l'infirmière). Faites en sorte de transmettre le numéro de l'infirmier depuis cette méthode vers la feuille de transformation comme indiqué dans le cours (voir aussi : Passages de paramètres XSLT en C#).

4 Projet Jeu vidéo

Liste des choses à faire :

- Réalisez en entier le jeu test `MyGame`
- Rédigez la fiche signalétique de votre jeu
- Réalisez les diagrammes UML de classe et de séquence