Rapport

Bordes Antoine , Duperier-Catalaz Mathis
Ben Touati Oumayma , Lagrut Baptiste

Rapport Jeux vidéo

Plan du Document

1. Introduction :

Ce document vise à présentser l'utilisation des différentes technologies XML et C# dans le cadre du projet de jeu vidéo et contient une description générale du projet et des besoins en termess de gestion des données. Il vise à expliquer comment différentes technologies sont utilisées pour gérer l'initialisation du jeu, les profils des joueurs, les sauvegardes des parties et la présentsation des données. Les technologies couvertes incluent les fichier XML, les schémas XML (XSD), les programmes C# pour la validation des fichier XML, les transformations XSLT, et diverses méthodes pour la manipulation et la sérialisation des données XML en C#. Chaque section détaillera le cas d'utilisation spécifique et fournira des exemples pour illustrer l'implémentation.

2. Fichiers XML :

On se sert de fichier XML afin de stocker toutes les données qu'on veut pouvoir changer facilement. Cela concerne diverses statistiques du joueur et des ennemis, que l'on veut pouvoir changer facilement pour équilibrer le jeu, mais aussi le niveau (incluant les collisions du niveau et les entités qui y sont présentses), les données d'animationss ainsi que les scores. A noter que toutes les données sont stockées dans le dossier *Content/Data*.

3. Schémas XML (XSD)

C'est simple, tous nos documents XML à l'exception des scores se conforment à un schéma. Cela nous permet notamment de vérifier au préalable tout accès à des données par une validation par rapport au schéma, ce qui nous assure de l'existence de tous les nœuds auquel on va vouloir accéder par la suite. Et cela nous évite par la suite de mettre de multiples conditions vérifiant chaque noeud.
Les schémas se trouvent dans le même dossier que les données qu'ils contraignent.

4. Programme C# de test de conformité et validité des documents XML

Cette validation est faite par les méthodes d'une classe static se trouvant dans le fichier `XMLUtils.cs`. Il s'agit des fonctions données dans le cours mais réécrites et complétés afin de les améliorer et de les coller à nos besoins. Ainsi on peut tout aussi bien effectuer la validation d'un fichier précis avec la méthode **ValidateXmlFile** que valider tous les fichier XML présents dans un dossier avec la méthode **ValidateXmlFiles**. Cela permet une validation simple et rapide. La classe permettant cela est static car cela nous permet de ne pas créé d'instance pour effectuer des validations. La plupart des validations se font dans Game1 à l'initialisation. Cela est surtout important pour les ennemis pour éviter de révalider plusieurs fois le même fichier.

5. Feuilles de transformation XSL

Nous nous servons de feuilles de transformation XSLT afin de traiter les scores. Pour notre jeu, un score classique n'aurait pas fait beaucoup de sens car il n'y a qu'un niveau qui se finiet vite. Nous avons donc décidé que l'objectif serait de finier le niveau le plus vite possible, et donc d'implémenter un timer. Ce timer s'arrête à la mort du boss, et le temps obtenu est sauvegardé dans le fichier `score.xml`. Ensuite nous avons chercher à extraire le meilleur temps (le plus faible donc). Pour cela, nous avons rencontré des problèmes, donc nous avons décidé de faire cette extraction en deux temps.

- On utilise une première transformation pour trier le fichier contenant les temps. Cela nous donne le fichier `sort_score.xml` avec les temps triés du plus court au plus long.
- On utilise ensuite une deuxième transformation pour récupérer uniquement le premier temps de cette liste, le plus court donc. Il se trouve dans le fichier `min_score.xml`.

Nous n'avons malheureusement pas d'autres transformations, l'explication est à la fin du rapport.

6. Utilisation de DOM + requêtes XPath

Dans notre projet, nous nous servons de DOM pour récupérer les données d'un fichier complet quand la sérialisation n'est pas une option viable. Concrètement, cela est le cas :

- Le traitement des niveaux de jeu
- Le traitement des animations

Niveaux du jeu

- **fichier** : `level1.tmx`, `Tile.cs`

Pour faciliter la création et la modification du niveau, nous avons choisi d'utiliser le logiciel Tiled. Ce dernier nous permet de peindre notre niveau, et ce sur différentes couches afin de pouvoir traiter certains éléments différemment. Une fois cela fait, ce logiciel nous permet aussi d'exporter toutes ces données dans un unique fichier XML (`level1.tmx`).
Notre niveau est divisé en 3 partie :

- Une partie destiné à être affiché. Cela correspond à toutes les plateformes et aux éléments d'arrière plan.
- Une partie contenant les collisions du monde, qu'on sépare car on veut les traiter mais pas les afficher. De plus tous les éléments ne sont pas nécessairement dotés d'une collision.
- Une partie entité, qui permet de placer les entités directement par le logiciel. Nous reviendrons sur ça dans la partie Sérialisation.

Quelque soit la partie, les informations sont stockés dans un dictionnaire qui associe un vector2 à une valeur. Pour comprendre son fonctionnement, partons d'un exemple. Le couple (2, 1) : 2 dit qu'à l'emplacement 2 *tileSize, 2* tileSize, on trouve la tile numéro 2 de la tilemap. Cela permet une traitement des données simples.

DOM nous sert à récupérer les différentes données du fichier de niveau. On récupère ainsi certaines valeurs nécessaires aux calculs, ainsi que les différentes couches (layer) de données du niveau. Il aurait été ici compliqué d'utiliser la sérialisation car quasiment toutes les données récupérés doivent être traités et non utilisées directement.

Animations

fichier :

- package Animation : `Animation.cs, AnimationManager.cs`
- tous les fichier du dossier *Content/Data/Animation*

Pour les données des animationss, nous avons créé nos propres documents XML afin de satisfaire nos besoins tout en restant simple et adaptable. Nos fichier contiennent :

- Les données communes à toutes les animations :
  - le nombre de colonnes du tileset des animationss, qui nous sert à faire les retours à la ligne quand l'animations est longue
  - la largeur et la hauteur d'une case d'animations
- les animationss décrites une à une avec son nom, son type (continu -> boucle, ponctuel -> une fois), et sa séquence d'index qui correspond aux cases d'animationss présentses dans le tileset.

De plus nos fichier sont nommés `[nom du tileset].xml` ce qui permet ensuite dans le code de déduire le nom du fichier de données d'animations et réduit le nombre de paramètres.

Les données de ces fichier sont lues par DOM et stockées dans les structures de données adéquates. Chaque animations est représentsé par une instance d'Animation, qui ne stocke que des nombres. Cela simplifie le traitement, et l'AnimationManager se contente de gérer les différentes animationss et de renvoyer la case d'animations à afficher.

7. Utilisation de XmlReader

XML Reader est utilisé uniquement pour lire rapidement une donnée unique. Cela est utilisé dans le projet pour récupérer le nombre de case par ligne dans le tileset des niveaux. Il s'agit d'un simple nombre se trouvant dans le fichier de donnée spécifique au tileset du niveau.
Cette opération se fait dans le fichier `Tile.cs`, dans la méthode *GetNumTilesPerRow()*.

8. Sérialisation avec XmlSerializer

La sérialisation est utilisée dans le projet pour charger les données du joueur et des ennemis. Ces classes se prêtent particulièrement bien à la sérialisation, car les données peuvent être utilisées directement sans avoir besoin d'être traitées. L'idée est ici de permettre la sérialisation d'un maximum de données clés afin de permettre un équilibrage du jeu. Cela comprend la vie des ennemis, mais aussi leurs fréquences d'attaque ou la taille de leur hitbox par exemple.

9. Autres points notables

Modularité

Dans ce projet, nous avons porté une attention particulière à essayer de garder un code clair, modulable et facilement extensible dans le futur. De plus, nous avons pensé nos données afin de permettre l'ajout rapide et facile de contenu. Nos fichier de données sont conçus pour être simples et facilement compréhensibles. Ainsi, ajouter les données d'un nouvel ennemi par exemple prend 10 minutes.

Care des modificateurs d'accès

Nous avons porté une attention particulière aux modificateurs d'accès afin de maintenir autant que possible l'encapsulation des données. Cela dit il s'agit aussi de maintenir un compromis afin que nous puissions facilement relire et utiliser le code des autres.

Un peu de fonctionnel

Enfin, nous avons essayé d'inclure un peu de style fonctionnel. Ces ajouts sont encore limités car cela a été fait à la toute fin du projet et nos connaissances sur ce sujet étaient

encore primitives. Mais on pourrÀ noter la fonction GetFramesArray, qui renvoie un tableau d'entiers contenant les index des frames de l'animations.

```
private int[] GetFramesArray(int numFrames, XmlNode animationsNode) =>
animationsNode.SelectNodes("frame")
                    .Cast<XmlNode>()
                    .Select(node =>
int.Parse(node.Attributes["numFrame"].Value))
                    .ToArray();
```

Conclusion et limites rencontrées

Ce rapport a détaillé l'utilisation de diverses technologies XML et C# dans le cadre du projet de jeu vidéo. Ce projet nous a permis d'exploiter et de mieux comprendre les différentes notions du cours.

Pendant ce projet, nous avons rencontré beaucoup de difficultés. De toute évidence, notre choix de projet était bien trop ambitieux. De plus, une gestion de groupe très discutable a encore un peu plus compliqué la réalisation.

Par conséquent, le projet rendu n'est pas complet. Bien que fonctionnel, le projet de base poussait un peu plus. Certaines classes présentses dans ce projet ne sont ainsi pas utilisées, à l'image de `Bubble.cs` qui servait à afficher une bulle de texte. L'idée de base était d'avoir des éléments narratifs, mais par manque de temps cette partie n'est pas finie. Il manque de plus certains éléments attendus dans le cahier des charges, comme par exemple la possibilité de sauvegarder, des transformations XSLT pour une meilleure gestion des hi-scores, et quelques éléments de sérialisation. Cependant toutes les notions ont été intégrées, et il en ressort un jeu incomplet mais quand même jouable.

Ce projet aura au final été très enrichissant, aussi bien en termes de code que de gestion de projet. Cette expérience acquise s'est déjà révélée très utile dans le projet intégrateur, et nous sera sans aucun doute précieuse à l'avenir.

Note : A cause d'un manque de connaissance avec Github, les commits présentss ne sont pas nécessairement représentsatif de la quantité de travail fourni par chaque personne du groupe. Ou, pour cité Maya, "we have angered a couple gods and now two groups members have no registered commits :C".