Maze Escape

Rapport technique – Projet Jeu Vidéo

Chapitre IV – MonoGame

Célia BOUAMARA
Khoyane DRAME
Sergiu MANEA
Filsane KASSIM ADAN

L3 MIAGE – 2025

Décembre 2025

Table des matières

1 Introduction

Dans le cadre du projet jeu vidéo, notre groupe a développé un jeu 2D intitulé **Maze Escape** avec le framework `MonoGame`. Le joueur incarne un astronaute qui doit sortir d'un labyrinthe en tirant sur des murs destructibles tout en évitant des fantômes qui le poursuivent. L'objectif principal était de mettre en œuvre les technologies demandées : XML/XSD/XSLT, parsers DOM, sérialisation, tout en respectant une architecture propre.

2 Utilisation des technologies demandées

2.1 XML et XSD – Chargement des niveaux

Les niveaux du jeu *Maze Escape* sont entièrement définis au moyen de fichiers XML externes, validés par un schéma XSD. Cette séparation entre données et code permet de modifier les niveaux sans changer l'application, ce qui rend le jeu extensible et facilement maintenable.

— **Fichiers de niveaux** : `maze1.xml`, `maze2.xml`
Chaque fichier XML décrit un niveau complet du jeu. Le moteur lit ces fichiers au lancement d'un niveau, ce qui permet de charger dynamiquement l'environnement.

```
            lignes="12"
            sortieX="18"
            sortieY="11">


    <!-- Position de départ du joueur -->
    <jeu:start x="10" y="5" />

    <!-- Grille du labyrinthe -->
    <jeu:grille>
        <jeu:row>1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1</jeu:row>
        <jeu:row>1,0,0,0,0,0,0,0,3,3,3,0,0,0,0,0,0,0,1</jeu:row>
        <jeu:row>1,0,1,1,1,0,1,1,1,0,0,1,1,1,0,1,1,1,0,1</jeu:row>
        <jeu:row>1,0,1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1,0,1</jeu:row>
        <jeu:row>1,0,1,0,1,1,1,0,1,1,1,1,0,1,1,1,0,1,0,1</jeu:row>
        <jeu:row>1,0,0,0,0,0,0,0,3,0,0,0,0,0,0,0,0,0,0,1</jeu:row>
        <jeu:row>1,0,1,1,1,0,1,1,1,0,0,1,1,1,0,1,1,1,0,1</jeu:row>
        <jeu:row>1,0,0,0,1,0,0,0,1,0,0,1,0,0,0,0,0,1,0,1</jeu:row>
        <jeu:row>1,1,1,0,1,1,1,0,1,1,1,1,0,1,1,1,0,1,0,1</jeu:row>
        <jeu:row>1,0,0,0,0,0,0,0,0,0,0,0,0,3,0,0,0,0,0,1</jeu:row>
        <jeu:row>1,0,0,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,1</jeu:row>
        <jeu:row>1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,2,1</jeu:row>
    </jeu:grille>

    <!-- Fantômes -->
    <jeu:ghosts>
        <jeu:ghost x="10" y="5" />
        <jeu:ghost x="15" y="1" />
    </jeu:ghosts>

</jeu:Labyrinthe>
```

— **Schéma XSD** : `maze.xsd`
Le schéma définit la structure exacte que doivent respecter les fichiers XML : types, attributs, éléments obligatoires, cardinalités, etc. Grâce à ce schéma, le XML est contrôlé et les erreurs de format sont détectées automatiquement.

— **Contenu du XML**
Chaque fichier de niveau possède les informations essentielles suivantes :

— **Dimensions du labyrinthe** : `colonnes`, `lignes`. Ces valeurs déterminent la taille de la grille chargée en mémoire.

— **Position de la sortie** : attributs `sortieX` et `sortieY`. Ils indiquent l'emplacement

exact où le joueur doit arriver pour terminer le niveau.
- **Point de départ** (`<Start>`) : coordonnées initiales du joueur, définies par les attributs `x` et `y`.
- **Structure complète de la grille** : élément `<Grille>` contenant plusieurs balises `<row>` représentant chaque ligne du labyrinthe. Chaque ligne contient une séquence de valeurs décrivant les murs, les passages libres et les murs destructibles.
- **Liste des fantômes** : élément `<Ghosts>` regroupant plusieurs `<Ghost>`. Chaque fantôme possède une position (`x`, `y`) définissant son emplacement initial dans le niveau.

— **Chargement dynamique**
Lorsqu'un niveau est lancé, le jeu lit automatiquement les fichiers XML :
- la grille du labyrinthe est reconstruite en mémoire,
- la sortie est positionnée,
- le joueur apparaît au point de départ,
- les fantômes sont générés via la lecture DOM du fichier.

Ce fonctionnement rend le jeu évolutif : ajouter un nouveau niveau consiste simplement à créer un fichier `mazeX.xml` compatible avec le schéma XSD.

2.2 Chargement XML : Désérialisation et DOM

Dans ce projet, je n'ai pas réalisé de sérialisation (sauvegarde) des données du jeu. En revanche, j'ai mis en place un système complet de **chargement XML** basé sur deux techniques différentes : la **désérialisation** pour le labyrinthe et le **DOM** pour les fantômes.

— **Désérialisation du labyrinthe**
Le fichier `mazeX.xml` contient la structure complète du niveau : dimensions (lignes, colonnes), murs, murs cassables et sortie. J'utilise une **désérialisation XML partielle** afin de reconstruire la matrice du labyrinthe (`int[,]`), qui sera utilisée pour la détection des collisions et l'affichage. La désérialisation permet d'obtenir facilement :
- le nombre de lignes et de colonnes,
- la position de la sortie,
- toutes les lignes de la grille (éléments `<row>`).

— **Chargement des fantômes avec DOM**
Les fantômes ne sont pas inclus dans la désérialisation, car leur liste est indépendante et structurée différemment. Pour cela, j'utilise un **DOM XML** (`XDocument` ou `XmlDocument`) afin de parcourir le document et d'extraire directement les nœuds `<ghost>`. Cette approche permet de récupérer les coordonnées (`x`, `y`) et de créer les objets `Ghost` dans le jeu.

— **Résumé des choix techniques**

- Labyrinthe : **chargé par désérialisation**, car structure régulière et adaptée au mapping en C#.
- Fantômes : **chargés via DOM**, car liste dynamique plus simple à parcourir de façon manuelle.
- Le jeu ne réalise **aucune sauvegarde**, donc il n'y a pas de sérialisation.

2.3 2.3 XSLT – Transformation

Objectif. Dans le cadre du projet, une feuille XSLT permet de transformer un fichier de sauvegarde XML en une **page HTML descriptive du jeu**. Cette page offre une vue synthétique et lisible des informations principales du labyrinthe.

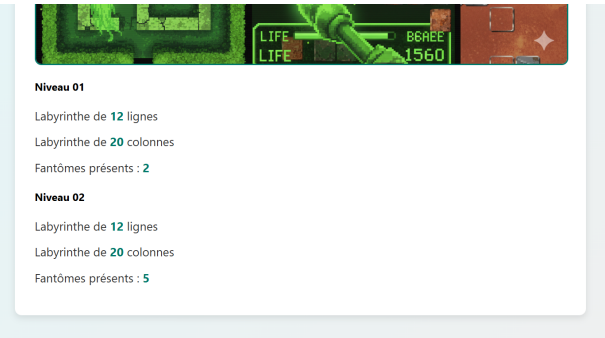
Fichier utilisé.
- **Feuille de style XSLT** : `report.xslt`
- **Entrée** : les fichiers décrivant un niveau, `mazeX.xml`

— **Sortie** : une page `.html` générée automatiquement

Contenu de la transformation. La transformation XSLT produit une page HTML contenant :

— le **nombre total de niveaux** disponibles dans le jeu ;
— les **dimensions du labyrinthe** (nombre de lignes et de colonnes) ;
— la **position du départ** et de la **sortie** ;
— la **liste des fantômes** avec leur position et leur état ;
— un aperçu textuel ou tabulaire de la grille du labyrinthe.

L'objectif principal est de permettre une **visualisation externe** du contenu du niveau, directement depuis un navigateur web, sans exécuter le jeu.



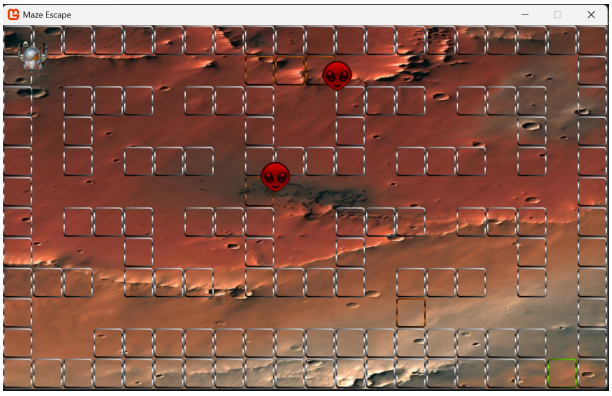
3 Étapes de développement de la partie C# via des techniques de programmation orientée objet

Le jeu a été développé en rajoutant des fonctionnalités pas à pas (l'approche n'a pas été globale dès le début). Des techniques de programmation orientée objet ont été employées afin de faciliter la lecture des classes (en particulier la classe `Game1`) et d'obtenir des classes indépendantes (éviter qu'un bug dans une classe rende le jeu injouable).

Les étapes principales ont été les suivantes :

1. **Sprites PNG** Utilisation de `mgcb-editor` pour importer les textures (personnage, fantômes, murs, balles). Textures chargées dans `Game1.LoadContent()` puis utilisées dans les méthodes `Draw()` des classes correspondantes.
2. **Setup .NET et MonoGame** Installation des deux environnements depuis les sites officiels.
3. **Changer le fond d'écran** Le fond est rendu dans `Draw()` via un fichier JPG. OOP : géré dans `Game1`, responsable du rendu global.
4. **Donner les dimensions de l'écran** Dans le constructeur `Game1` : _graphics.PreferredBackBufferWidth = 1280 ; _graphics.PreferredBackBufferHeight = 768 ; OOP : encapsulé dans `Game1`.
5. **Changer le fond d'écran en Mars** Remplacement du fond dans `LoadContent()`. OOP : même classe.
6. **Ajouter le sprite protagoniste** Chargement du sprite et création de l'objet `Personnage`. OOP : la classe `Personnage` encapsule la position et le mouvement.
7. **Fonction de mouvement (flèches → déplacement)** Implémentée dans `Personnage.Update()` via `KeyboardState`. OOP : encapsulé dans la classe `Personnage`.
8. **Limites de l'écran** Ajout de `Clamp` dans `Personnage.Update()`. OOP : responsabilité du personnage.

9. **Labyrinthe (niveau 1) et murs destructibles** Classe `Labyrinthe` utilisant une grille `int[,]` et gérant affichage + collisions. OOP : séparation nette du décor et de la logique du joueur.
10. **Classe Bullet (tir + destruction de mur)** Classe indépendante gérant mouvement et collisions. OOP : encapsulation du comportement de la balle.
11. **Classe Ghost (déplacement autonome, collision dangereuse)** La classe gère : – déplacement autonome, – contact = Game Over, – disparaît après 3 tirs. OOP : encapsulation de l'entité ennemie.
12. **Deuxième niveau** Chargé via `LoadLevel`. OOP : réutilisation complète de l'architecture.



4 Gestion de projet

L'organisation a été difficile : le groupe n'était pas structuré, certain·e·s membres avançaient seuls et une personne a même abandonné en cours de route. Cela a nécessité une réorganisation complète de notre manière de travailler.

— **Répartition révisée des tâches** après le départ d'un membre afin d'équilibrer la charge de travail.
— **Mise en place tardive mais efficace d'une organisation Git** :
— **Approche itérative progressive** : d'abord une version console minimale, puis intégration de MonoGame et ajout des fonctionnalités (collisions, sprites, fantômes, etc.).
— **Tests fréquents** réalisés au fur et à mesure pour stabiliser le gameplay et éviter les régressions.

Malgré le départ d'un membre et les difficultés initiales, nous avons quand même réussi à rendre un travail, car un travail prévu pour quatre personnes peut être réalisé par trois, voire même par deux.

5 Analyse a posteriori

5.1 Ce qui a bien fonctionné

— Séparation parfaite données/code grâce à XML
— Chargement dynamique des niveaux et fantômes
— Sauvegarde fonctionnelle et simple à étendre
— Respect des consignes techniques

5.2 Difficultés rencontrées

— Problèmes de chemins avec espaces dans les dossiers (OneDrive)

— Première utilisation de MonoGame Content Builder

5.3 Améliorations possibles

— Validation XSD en runtime
— Utilisation de `XmlSerializer` au lieu de sérialisation manuelle
— Ajout d'un menu avec Myra
— Le jeu revient au premier niveau après la fin du niveau 2
item Sauvegarde de la position du joueur et du niveau courant

6 Conclusion

Ce projet nous a permis de maîtriser :
— La séparation données/logique via XML
— Le parsing moderne avec LINQ to XML
— La sérialisation et la persistance
— Une architecture propre et extensible

Le jeu est fonctionnel, les niveaux sont modifiables sans toucher au code, et les technologies demandées sont implémentées de manière claire et efficace.