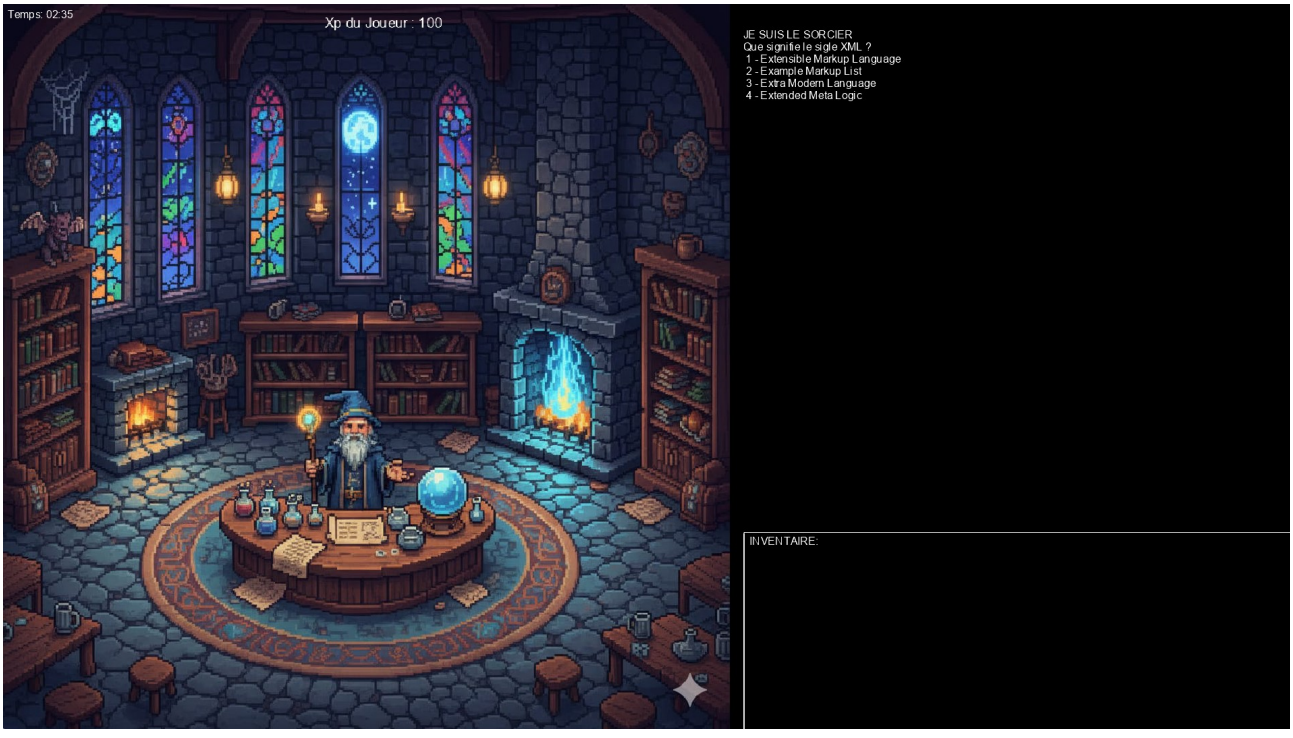
Analyse & Explications du Projet C#/XML
Expédition au royaume médiévale

Pierre Médina
Maxence Theron
Donatien Fofana
Mamadou Diallo

Rendu final :

Une fois un portail passé :



Répartition du travail :

Un chef de projet pour tout: Pierre Médina
Un chef adjoint spécialisé en Xml: Fofana Donatien

Répartition du travail toujours par équipe de 2
Avec deux membres polyvalents: Maxence & Mamadou

Documentation Technique :

Manipulation du Jeu :
Fichier : **README.txt**

(aussi disponible dans le README du fichiers zip Jetbrains)

Pour ce jeu, la **taille de la fenêtre** de l'écran se change dans
Globals, il est conseillé de jouer en 1920 (Width) * 1080 (Height) ou en 1000*1000

Mais si cela n'est pas possible le jeu devrait normalement s'adapter pour un écran plus petit (nous reviendrons sur cet aspect plus tard).

Pour **Sauvegarder** (par Serialization & ParserDOM), il est nécessaire de mettre échap pour quitter le jeu ! La sauvegarde se fait automatiquement sinon.

A noter : certains éléments du jeu sont sauvegarder (vie, xp, les deux objets obtenus chez le mage, les infos de combat... Mais pas tous. Ce n'est pas le cas de l'épée par exemple).

Bonne partie !

Échanges :
Nous avons procédé par trois types d'applications :
Essai sur Github - Serveur Discord

Mais c'est sur notre Groupe WhatsApp que nous étions le plus actifs (un point sur le projet, au moins une fois chaque semaine)

Explication Générale du C# :

Class Player :
Fichier : **Player.cs**

Ajout d'un chiffre coordonnées à chaque fois qu'on appui sur les boutons Up, Down, Left et Right. Le tout multiplié par une vitesse définie à la base du player (voir _player2 dans Game1).

Spécificité de la Class : Vérifie que les coordonnées suivantes pour X ou Y sont valides et empêche au joueur de sortir de la carte.

Class Lieu – Spécifié du projet :
Fichier : **Lieu.cs**

Cette class se décompose en 3 méthodes principales dont une (Drawlieu) plus volumineuse.

- DrawPortes : Dessine la «porte» (une image, ici un portail) sur la carte de base
- VerifierContact: Vérifie si le joueur entre en contact avec l'image, considère les images comme deux rectangles
- DrawLieu : Dessine l'intérieur d'un Donjon, avec une Image et des rectangles noirs, une partie inventaire.

Il y a également la méthode UpdateLieu pour gérer les questions réponses sur Xml du sorcier.

Détails des spécificités de DrawLieu : cette class commence par calculer en attribuant 75 % de l'espace à l'image comment l'imprimer dans le plus grand espace possible (sans déformation) . Pour cela elle fait le calcul si l'impression se fera en fonction de la largeur ou de la hauteur.

Elle complète ensuite si nécessaire par deux rectangles noirs, en vérifiant si l'image imprimée prend toute la largeur ou hauteur. L'impression du texte peut donc se faire à droite ou en bas (dépend du bool haut), c'est en cela que le jeu « s'adapte » à la taille de l'écran.

Class Game1 :

Fichier : **Game1.cs**

Réalise les impressions de base.
Ainsi que les Uploads qui revoient vers des ifs dans Lieu à chaque appui de touches.

Pour ce qui est des impressions de portes et de lieux, cela fonctionne par déclaration d'une liste d'objets :

Déclaration des lieux de type « Lieu »

```
_lieux = new List<Lieu>();
_lieux.Add(new Lieu( nom: "auberge", xHaut, yHaut, width: 128, height: 128, porte: _portes, imageLieu: _auberge, _xp, lieu_num: 1, texture_bonus: _auberge, event_num: 0));
_lieux.Add(new Lieu( nom: "sorcier", xGauche, yGauche, width: 128, height: 128, porte: _portes, imageLieu: _sorcier, _xp, lieu_num: 2, texture_bonus: _sorcier, event_num: 0));
_lieux.Add(new Lieu( nom: "monstre", x: xbas, y: ybas, width: 128, height: 128, porte: _portes, imageLieu: _monstre, _xp, lieu_num: 3, texture_bonus: _monstreB, event_num: 0));
_lieux.Add(new Lieu( nom: "king", xDroite, yDroite, width: 128, height: 128, porte: _portes, imageLieu: _king, _xp, lieu_num: 4, texture_bonus: _king2, event_num: 0));
base.Initialize();
```

Puis impression à l'aide d'un for :

```
// Impression portes
for (int i = 0; i < _lieux.Count; i++)
{
    _lieux[i].DrawPorte(_spriteBatch);
}
```

```
// Images des lieux (par dessus tout si contact)
_spriteBatch.Begin( sortMode: default, blendState: null, SamplerState.PointWrap);
for (int i = 0; i < _lieux.Count; i++)
{
    _lieux[i].DrawLieu(_spriteBatch);
}
_spriteBatch.End();
```

(il est donc possible d'avoir un portail spécifique pour chaque Donjon, mais cela n'a pas été utilisé)

Le Game1 effectue également la Charge/Sauvegarde (mais cette partie sera de nouveau détaillée dans l'explication de la Serialization) :

```
//charger donnees
Jsauvegarde = new XmlSerializerJeu();
Jsauvegarde.Charger();
```

Sauvegarde à l'exit

```
if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed ||
    Keyboard.GetState().IsKeyDown(Keys.Escape))
{
    Exit();
    //sauvegarder donnees
    Jsauvegarde.Sauvegarder();
}
```

Documentations Explicatives:

Avantages du projet :
Comme dit précédemment, la fenêtre s'adapte aux types d'écran, cela a ajouté de la complexité au projet mais selon moi, c'est un super atout qui permet d'imprimer l'image sans la déformer et donc de maintenir un jeu agréable.

Le projet est accessible et compact, pas de complexité inutile, tout est fait avec trois classes, ce qui le rend facile à comprendre en globalité).

Échecs :

Il y à eu une tentative d'animer le « player », qui était trop chronophage et qui a donc été annulée pour se concentrer sur l'essentiel.

Mauvaise connaissance de départ de Game1 qui entraîne une classe "lieu" lourde et des liens superflus. La méthode DrawLieu est beaucoup trop lourde, et n'a pas était divisés par manque de temps.

Conclusion Gestion de projet :
Nous aurions dû nous distribuer les différentes parties du projet dès le départ . Nous ne l'avons pas fait par manque de connaissances sur comment faire un jeu de ce type. En conclusion on à pas passer assez de temps à la conception en C# et cela nous a desservi.

Utilisation de XmlReader :
Fichier : **JeuXmlReader.cs**

Lit tout le Xml et récupère les questions et les renvois, en mettant « 0 » entre chaque questions.
Le but de cette procédure et de lire/récupérer qu'une question par une, elle relève de l'astuce mais fonctionne bien ici (la version tableau n'a jamais marché).

Utilisation de la sérialisation :
Fichiers : **JoueurSauvegarde.xml - JoueurSauvegarde.xsd – XmlSerializer – JoueurSauvegarde.cs – Game1**

Sérialise/Désérialise avec les fonctions Sauvegarder()/Charger(), en utilisant ChargerData() et RestaurerDonneesDansJeu() deux méthodes de JoueurSauvegarde (elles-mêmes dépendantes d'attributs de Globals).

XSLT :
Fichiers : **Jeu.xslt - jeuXsd.xsd – XmlJeu.xml – JeuXslt.html**
Une transformation en page html est effectuée sur le Xml du Jeu (et non pas celui de JoueurSauvegarde), visible dans le dossier html.

début du html :

Caractéristiques du Joueur

Nom: LéoP12

Vie: 150

Énigmes du Donjon

Question : Que signifie le sigle XML ?
1 - Extensible Markup Language
2 - Example Markup List

ParsersDom :
Fichiers : **DataCombat.xml – DataCombat.xsd – JeuDomPath - Game1**
Effectue une sauvegarde en récréant le xml. La sauvegarde se charge/conserve de la même façon que le Serializer à la différence près que nous avons essayé de compacter les données et Dom dans en même fichier et que celle-ci utilise Xpath.