RAPPORT SPACEPEACE

ROUDOCI Reza
TASSAUX Aurélien
JEANNOT Krys
NICOLAS Juliette

Table des matières

I) Introduction

1) Objectif du projet

L’objectif de ce projet est de concevoir un jeu vidéo sous MonoGame en C# afin de mettre en pratique toutes les techniques étudiées en cours et mise en pratique en TP. Il nous a permis de développer nos compétences en programmation orientée objet. Ce projet a également pour objectif de nous tester et d’apprendre les tenant et aboutissants du travail de groupe

2) Contexte

Le jeu se décompose en plusieurs sous-dossiers. Le C# pour créer les classe du jeu, le xsd pour définir les modèles de différentes classes serializable, le dossier xml pour avoir les fichiers xml qui inscrit des valeurs grâce au schéma xsd, le dossier xslt pour avoir des transformation xslt afin de créer dans un dossier html des fichiers html. Sont également présents les différents sprites et tuiles nécessaires à l'apparat du joueur et des niveaux de jeu. Ainsi que le mgcb permettant de tous les intégrer au jeu.

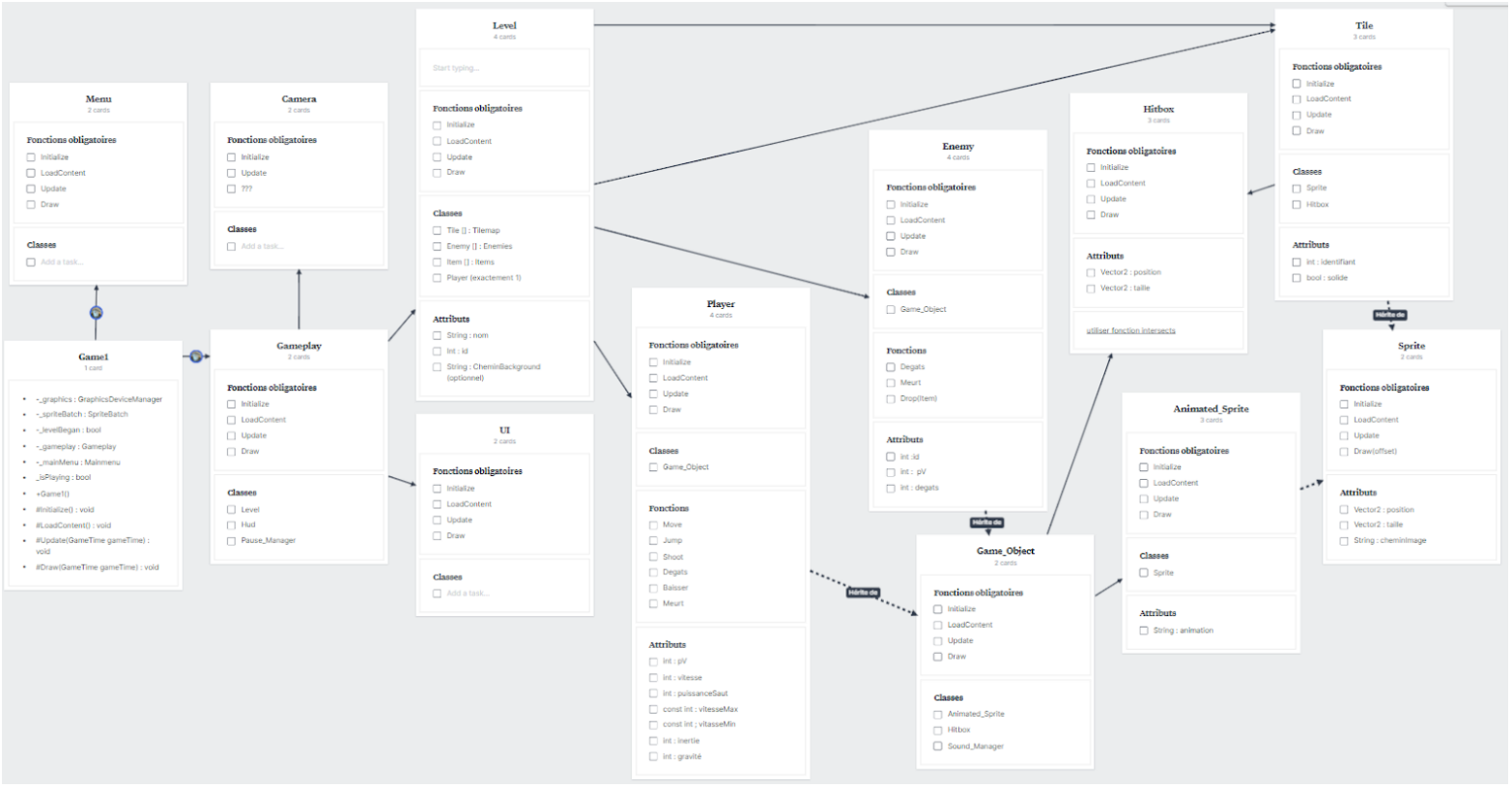
3) Organisation

Nous avons organisé notre projet en nommant un chef de groupe Roudoci Reza afin d’avoir une bonne organisation, de coordonner les différentes tâches et garantir une bonne synchronisation. Ce choix a été motivé car il s’agissait de notre seul membre ayant une expérience précédente avec le développement de jeux vidéo.
Afin d’optimiser notre travail nous avons utilisé le site GitHub pour gérer et partager notre travail avec tous les membres du groupe.
Voici nos identifiants sur GitHub :
Roudoci Reza: rzmobile / BuildTools
Nicolas Juliette: julietteNi / nicolasj / nicolas
Tassaux Aurélien: Grimoire394 / Aurélien
Jeannot Krys: krukry05 / krukry / jeannokr / krysjeannot

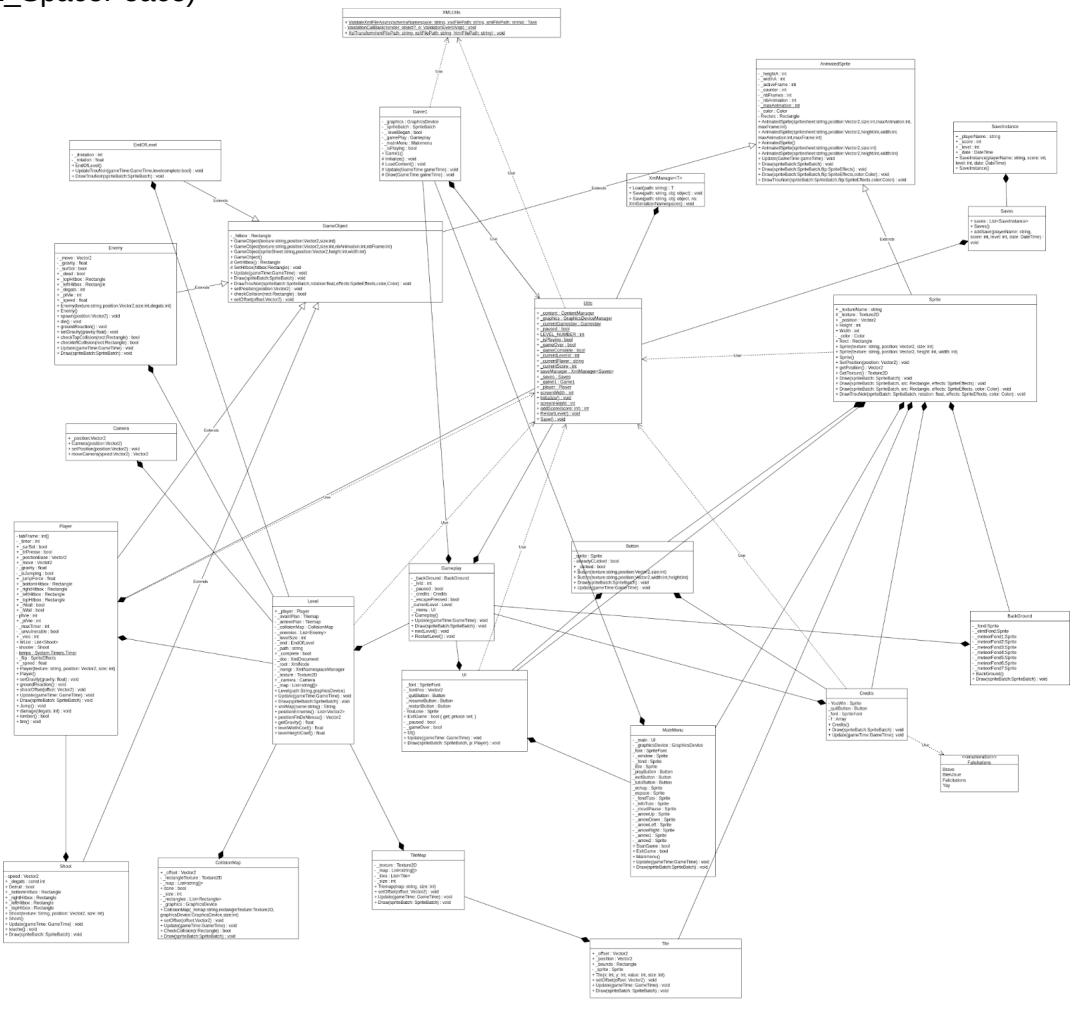
4) Bref résumé du jeu

Notre jeu s'appelle SpacePeace, c’est un jeu de plateforme 2D où le joueur incarne un astronaute. Armé de missiles, il peut éliminer les ennemis.
Le jeu se contrôle avec les flèches directionnelles clavier ainsi que la souris pour naviguer dans les menus de pause et principal.

II) Modélisation UML



Ceci est la première version de l'UML de notre platformer. Nous l'avons depuis modifié car il était bien trop ambitieux. En voici la dernière version. Disponible en image (UML_SpacePeace)



III) Réalisation technique

1) Explications de nos choix

- Déplacement du joueur

Nous avons choisi de permettre au joueur de se déplacer horizontalement pour qu'il puisse parcourir la map, et à la verticale pour qu'il puisse sauter les obstacles ou encore tuer les ennemis.
Cependant, nous avons pris la décision de ne finalement pas pouvoir faire de niveau vertical, car cela aurait impliqué de faire bouger la caméra à la verticale. C'est-à-dire une caméra qui suit le joueur lors de ses sauts cela aurait rendu l'expérience de jeu trop fluide et pouvait devenir très désagréable à jouer.

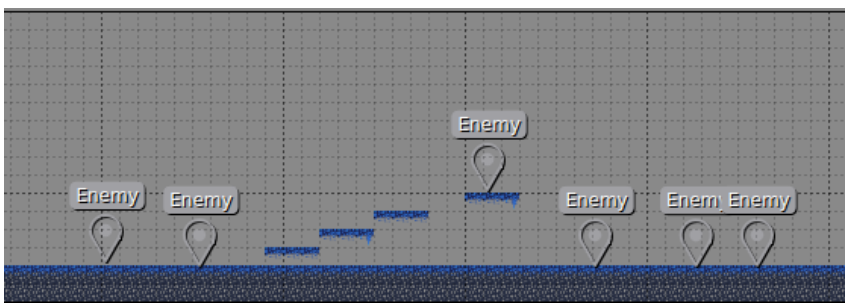
Nous lui avons attribué une variable de vitesse, il ne peut pas accélérer, car nous avons choisi la solution la plus simple possible pour commencer. Par la suite, nous n'avons pas trouvé pertinent dans le cas de notre jeu d'avoir une accélération sachant qu'il n'y a pas de temps pour chronométrer le temps du joueur.

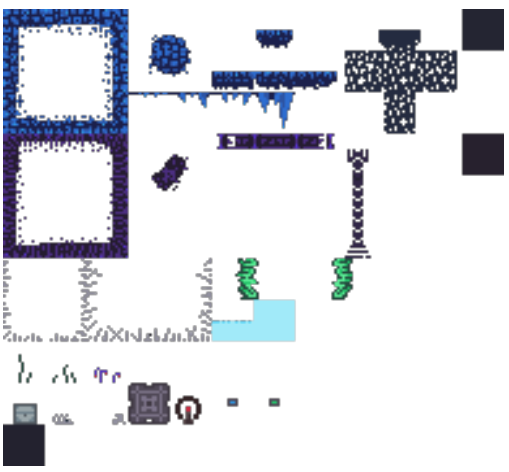
- Tire de missile

Un missile est un objet qui a une vitesse, un nombre de dégâts infligés lors d'une collision et un booléen permettant savoir s'il est détruit. Dans la classe Player, une liste de tirs est maintenue. Dès que le joueur appuie sur la touche Espace, on ajoute une instance de tir à cette liste. Pour chaque tir de la liste, nous vérifions s'il n'est pas détruit. Si c'est le cas, nous le faisons avancer et l'affichons à l'écran.
Nous avons choisi cette approche car elle nous permet de gérer plusieurs tirs simultanément sur une même image.

- Logiciel pour générer des maps

Nous avons conçu nos maps à l'aide du logiciel Tiled, car il génère des fichiers XML compatibles avec notre jeu. Nous avons sélectionné un tileset en adéquation avec le thème spatial de notre jeu, et positionné les tuiles sur une map pour créer des niveaux personnalisés.
Afin de structurer correctement les niveaux, nous avons créé trois calques distincts : Collision, AvantPlan et ArrierePlan. Collision ne s'affichant pas dans le jeu, nous avons donc créer le design du sol et des objets à sauter grâce au calques AvantPlan. Le calque ArrierePlan nous à permis de pouvoir mettre des décors en fond.





- **Gravité**

Nous avons conçu la gravité grâce à l'application de la variable de gravité à la vélocité verticale du joueur constamment. Quand la collision du bas du joueur s'intercepte avec la collision du sol, nous lui appliquons l'opposé de cette variable de gravité afin que le joueur s'arrête au niveau du sol et ne puisse pas le traverser.

- **Fin du Jeu**

Un niveau se termine lorsque le personnage rentre dans le trou noir. Nous avons choisi de faire finir un niveau lorsqu'il rencontre un objet car c'était plus facile pour nous. Si ce n'est pas le dernier niveau, le niveau suivant se lance directement,sinon, l'écran de fin s'affiche.

- **Ennemis**

Nous avons mis une seule sorte d'ennemis dans notre jeu par manque de temps pour en concevoir d'autres. Ils apparaissent uniquement lorsque leurs points de départ se trouvent dans l'écran. Nous avons décidé de faire ceci pour optimiser le code et qu'il n'ai pas à faire tous les calculs de mise à jour de l'ennemi tout le long de la partie. Mais également pour avoir des ennemis tout le long du niveau et qu'ils ne soient pas tombés dans un trou avant qu'on arrive au point où ils sont apparus, ou qu'on ne les ait pas tués en lançant beaucoup de missiles d'un coup en début de partie.

2) Présentation du code

- **Feuilles de transformation XSLT (pour 2 HTML, XML)**

Nous utilisons une fonction C# pour faire générer un fichier XML de sauvegardes
Une feuille de transformation XSLT qui prend ces fichiers de sauvegarde et convertit en une page HTML (dans un tableau) en triant par ordre décroissant du score.

- **Utilisation du DOM + requêtes XPath**

Nous avons utilisé du DOM pour générer des niveaux. La procédure se sépare en 2 parties: la classe Level récupère les chaînes de caractère correspondants aux niveaux dans le fichier XML à partir d'une requête XPath. Et ceci pour chaque calques.

Level récupère la position des ennemis et la fin du niveau.
Les classes TileMap et CollisionMap, récupèrent leurs chaînes de caractères correspondante, puis les sépare en une matrice et lit chaques valeurs de celle-ci pour lui affilier la taille ou la collision correspondante.

Nous avons procédé comme ceci pour des raisons de clarté du code.

- **XMLReader**

On utilise XMLReader pour charger la sérialisation. Ainsi que pour la validation des XML avec leurs schémas.

- **Sérialisation**

On sérialise les caractéristiques des ennemis et du joueur dans un fichier XML. Ceci nous permet d'avoir des statistiques modulables.

- **Listes**

Nous avons une liste de shoot dans la classe player, liste de Tiles dans la classe TileMap, et une liste de rectangle dans la classe Collision Map et une liste d'ennemis dans Level.

- **Énumérations**

Nous avons fait une énumération pour la fin du jeu. Lorsque le joueur finit tous les niveaux, il reçoit un message aléatoire de félicitation parmi 4 messages distincts. Nous avons fait cette énumération pour que le joueur puisse avoir des messages différents lors de chaque victoire.

- **Données en lecture seule et d'autres en lecture-écriture**

Lecteur seul : AnimatedSprite, on ne peut pas écrire RectSrc, il n'y a pas de set et il est défini automatiquement par d'autres valeurs. La frame actuelle, la taille que doit faire le Sprite, le nombre d'animation,...
Également dans Sprite, rectangle destination est défini par d'autres valeurs, la position et la taille. Le nombre de niveau est codé en dure dans la classe Utils

Lecture-écriture : Le score, l'ID du niveau actuel et le nom du joueur entre autres. Pour la sauvegarde, pour pouvoir modifier ces valeurs en temps réel.

IV) Conclusion

1) Bilan du projet

Finalement, concevoir un jeu vidéo est bien plus complexe que ce que nous avions pensé au premier abord. Il y a beaucoup plus de choses à penser que ce que nous imaginions. Mais ce projet nous a permis d'améliorer nos compétences en programmation orientée objet et de renforcer notre capacité à gérer le travail en équipe.

2) Perspectives d’évolution

Nous avions envisagé plusieurs améliorations pour enrichir le jeu mais nous n'avons malheureusement pas eu le temps de faire.
Il y avait principalement l’intégration d’effets sonores, comme des musiques de fond ou même des sons associés aux actions tels que les mouvements ou les tirs. Nous voulions également ajouter des items à collecter ou utiliser dans les niveaux, tels que des points de vie à gagner,... . La diversité des ennemis était également un point que nous aurions voulu faire avec des comportements variés pour intensifier le gameplay.
Nous aurions également voulu intégrer un delta time afin de fluidifier davantage les mouvements.
Pour finir, le dernier point d'amélioration était la diversité des niveaux. Nous avions comme projet de faire un niveau par planète, et donc de choisir différentes tilesets, fond, sons, luminosité,... pour chaque niveau. Comme par exemple :

- Planète de Feu : Présence de geysers de lave
- Planète de Glace : glissant rendant les déplacements plus difficiles
- Planète de Gravité Faible : Saut plus longs, difficulté à contrôler le personnage

Les niveaux devaient donc devenir progressivement plus sombres, oppressants et surtout de plus en plus compliqués.