



Rapport de Projet - C++ - Pôle Informatique

Rédigé par :

Erwan BOUVART
Fabien GRONDIN
Mathieu PAILHE
Valentin ROY

PROJET C++ **JEU VIDÉO MARIO LIKE**

Table des matières

1) Présentation du projet	1
2) Diagramme UML	2
3) Fonctionnalités principales	3
4) Gestion du projet	6

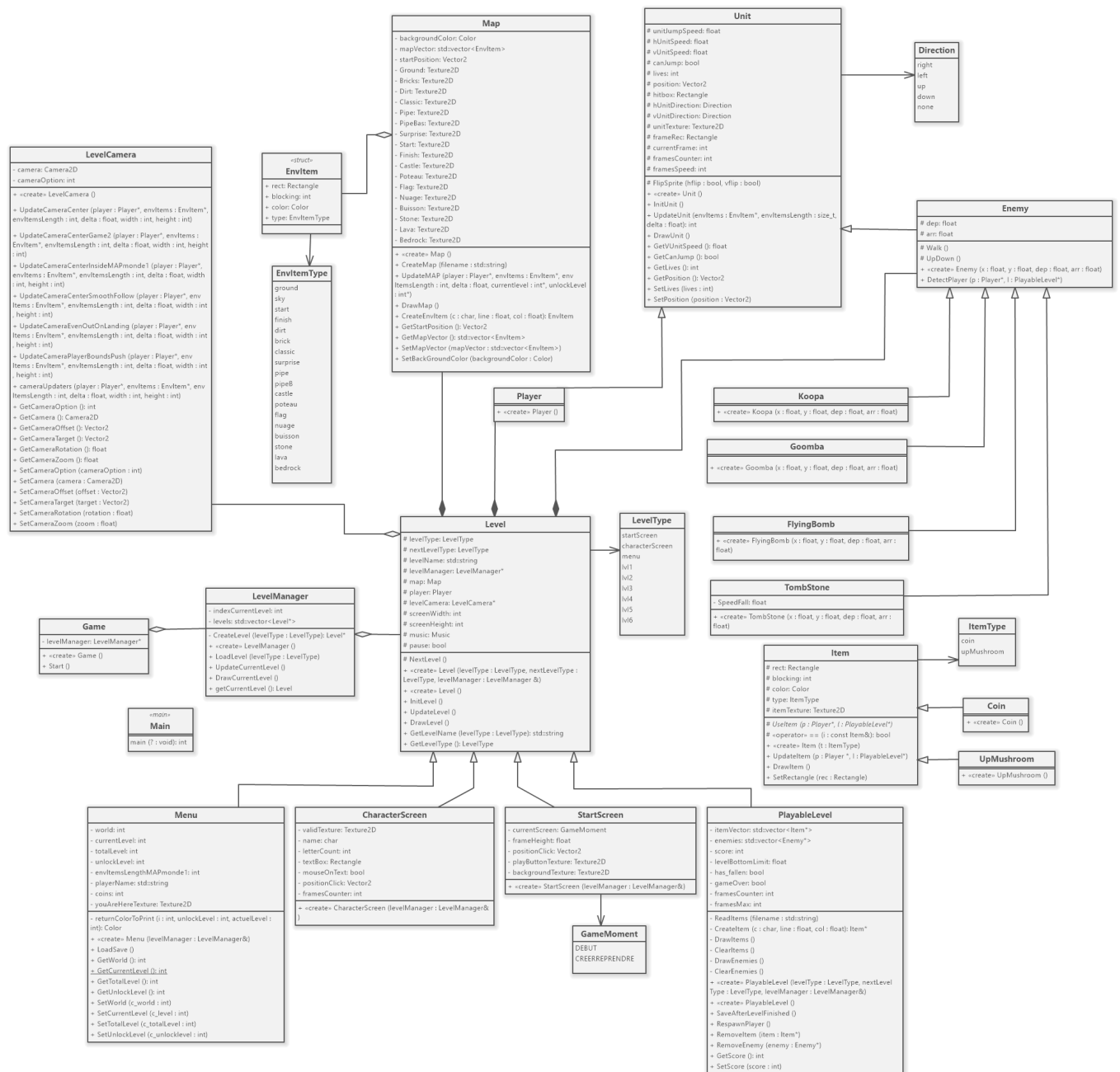
1) Présentation du projet

L'objectif de ce projet de C++ était de développer un logiciel pouvant être un "Casual Gaming", mettant en application les principes de Programmation Orientée Objet que nous avons étudiés lors de nos cours de C++.

Nous avons choisi comme jeu, un "Mario like", plus précisément un "Mario Bros" qui est un jeu d'arcade en 2D.

Notre jeu possède des écrans de démarrage, de création de partie et un menu permettant d'accéder à 6 niveaux, débloqués au fur et à mesure. Chaque niveau possède sa propre map, un joueur, des ennemis ainsi que des objets à collecter.

2) Diagramme UML



Le diagramme est aussi disponible dans le git du projet, au format png ou simp.

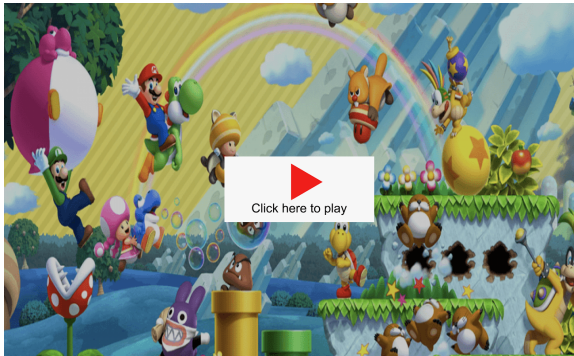
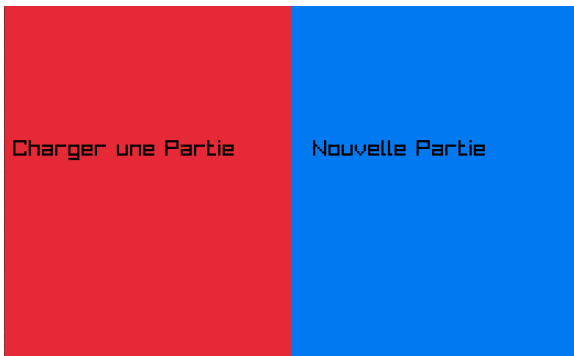

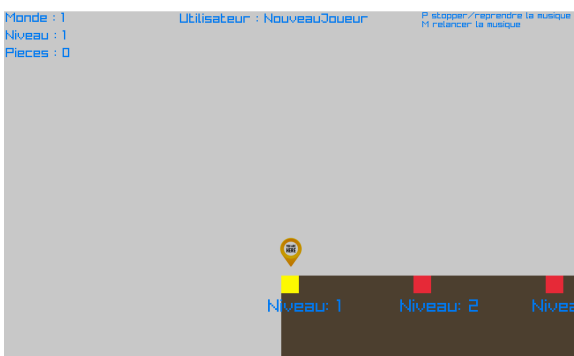
3) Fonctionnalités principales

Nous avons respecté les contraintes de code demandées explicitement dans le sujet.

Voici ce qui a été implémenté. Notre code :

- est commenté.
- respecte une convention de nommage.
- utilise la librairie externe Raylib pour l'interface graphique, l'animation et l'audio.
- respecte l'encapsulation. Tous les attributs sont privés ou protégés. Seules les méthodes qui le nécessitent sont publiques (généralement constructeur/getter/setter)
- possède 18 classes.
- possède 2 niveaux d'héritage. En effet, les classes Player et Enemy héritent de Unit et les classes Goomba, Koopa et FlyingBomb héritent d'Enemy.
- possède du polymorphisme. Par exemple, la fonction UseItem() définie dans la classe Item est override et possède une implémentation différente dans ses classes filles (Coin et UpMushroom)
- utilise l'enregistrement sur fichier texte pour sauvegarder la progression du joueur. Cet enregistrement se fait avec les fonctions Save et LoadSave, respectivement dans les classes PlayableLevel et Menu.
- possède une surcharge de l'opérateur == dans la classe Item, pour vérifier l'égalité entre 2 objets.
- utilise des vecteurs, notamment pour stocker les ennemis et les objets des niveaux.
- est disponible sur GitHub avec des commits réguliers et explicites.

Fonctionnalités du jeu :

<p>Ecran de démarrage (écran titre)</p>	
<p>Cet écran permet de choisir entre charger une partie déjà enregistrée, ou créer une nouvelle partie.</p> <p>Cliquer sur Charger une Partie vous fera passer directement au menu de sélection des niveaux.</p>	
<p>Pour créer une partie, il suffit d'entrer un nom de joueur, et de cliquer sur l'icône de validation.</p> <p>La partie sera automatiquement enregistrée dans un fichier texte qui sauvegardera la progression du joueur.</p>	
<p>Dans le menu, on voit les informations de progression du joueur, telles que le nombre de pièces collectées et les niveaux débloqués ou non (cases vertes ou rouges).</p> <p>L'utilisateur sélectionne un niveau disponible avec les flèches et appuie sur la touche entrée pour le lancer.</p>	

Notre programme génère les différents blocs et items du niveau automatiquement, à partir de 2 fichiers définissant la position et le type des objets et des blocs (map).

Nous avons 2 types d'objets, 4 types d'ennemis et une dizaine de textures de blocs.

Dans les niveaux, le joueur et la caméra commencent toujours sur le bloc de départ du niveau (bloc doré) et le joueur doit atteindre le bloc de fin (bloc doré).

Au niveau des collisions, le joueur vérifie la collision avec les blocs d'environnement proches. Les ennemis et les items vérifient la collision avec le joueur. Pour tuer un ennemi, il faut sauter sur sa tête. La collision avec des ennemis (ailleurs que sur leur tête) et les chutes retirent une vie au joueur.

Pendant le jeu, le joueur voit son score, ses vies et un décompte de temps, non bloquant, indiquant juste la durée normale du niveau.

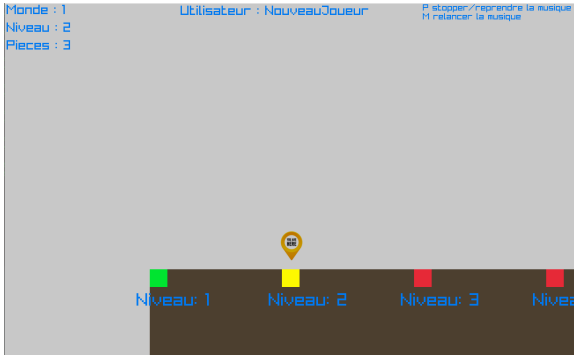
De plus, nous avons implémenté l'animation du joueur et des ennemis, et ajouté une musique au niveau.

Nous avons ajouté des fonctionnalités affichées en haut, à droite de l'écran, telles que : réapparaître au début du niveau, retourner au menu, arrêter la musique...

Lorsque le joueur a 0 vie et perd encore une vie, c'est "Game Over" et on est redirigé au menu de sélection des niveaux.

Si on atteint la fin du niveau, le niveau suivant se lance automatiquement.



<p>A chaque retour au menu, on charge le fichier de progression du joueur. Ce fichier est mis à jour uniquement lorsque le joueur termine un niveau, en touchant le bloc d'arrivée.</p> <p>Dans le cas du "Game Over" ou du retour manuel au menu, le fichier de progression n'est pas mis à jour (perte des objets collectés).</p> <p>Dans l'image de droite, on voit notamment que le niveau 2 s'est débloqué et que le nombre de pièces collectées a augmenté, suite à la complétion du niveau 1.</p>	
----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------

4) Gestion du projet

Tout d'abord, en ce qui concerne la communication, nous avons utilisé Discord pour discuter du projet et faire nos réunions, et GitHub pour partager et imbriquer nos codes.

Nous avons scindé notre temps en deux parties. Dans un premier temps, la période avant les partiels nous a permis de nous familiariser avec "Raylib" et nous avons pu commencer à réaliser notre projet, notamment la structure de la solution (premiers designs UML), le menu et un début de map simple.

C'est après les partiels que nous avons réellement commencé la programmation de notre jeu, avec toutes ses fonctionnalités. Nous avons suivi la méthode Agile avec des réunions régulières, des sprints de 3 jours en moyenne et un tableau de suivi des tâches (Google Docs).

Nous avons réussi à réaliser ce projet, en nous répartissant le travail, tout en restant en communication constante :

- Mathieu a travaillé sur les différents écrans de démarrage, sur la création de partie, sur la map (avec l'aide de Fabien), sur le menu et sur la sauvegarde de progression.
- Fabien s'est occupé des fonctionnalités du joueur, des items, du gestionnaire de niveaux, des niveaux, de l'animation (avec Valentin), de l'audio et du design du niveau 6.
- Valentin a traité les ennemis et leurs collisions avec le joueur (avec Fabien et Erwan), les hitboxs, l'animation des ennemis et le design des levels.
- Erwan a été l'électron libre qui a aidé dans les différentes parties, notamment les ennemis et leurs collisions.