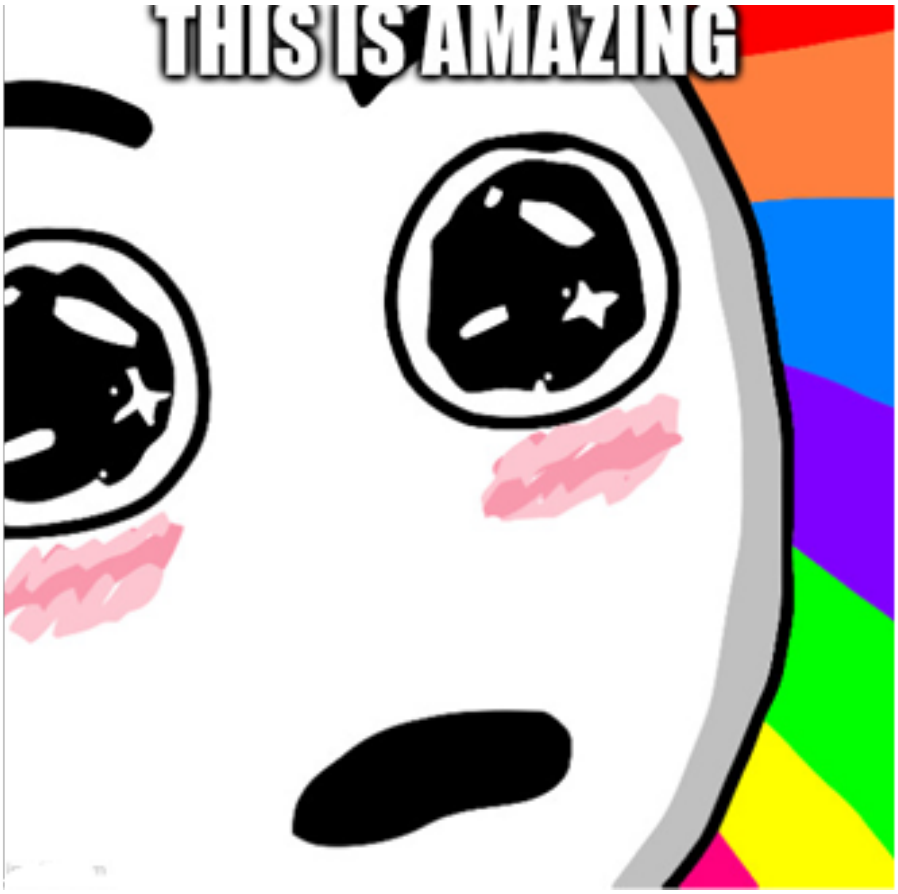
# Projet C++

Leo Porcher, Anes Abdou

Janvier 2025



# Contents

# 1 Introduction

## 1.1 Objectif du projet

L'objectif principal de notre projet est de concevoir et développer un jeu qui incarne pleinement l'esprit du concept "C'est incroyable". Afin de concrétiser cette vision, nous avons choisi de créer un jeu de plateforme, un genre ludique que l'on pourrait comparer à la référence qu'est **Mario Bros**.

Ce jeu proposera au joueur une expérience captivante, alliant des mécaniques de jeu dynamiques, des graphismes soignés et une jouabilité fluide. L'objectif est de créer un univers immersif qui saura captiver et surprendre les utilisateurs, tout en respectant les standards de qualité et d'innovation du domaine.

Nous avons pris soin de respecter un ensemble de contraintes techniques et fonctionnelles imposées par le cahier des charges, ce qui nous a permis d'adopter une approche méthodique et structurée tout au long du processus de développement.

## 1.2 Contraintes respectées

Fais une liste rapide des contraintes techniques que tu as respectées (ex : 8 classes, 3 niveaux de hiérarchie, etc.).

Pour convenir au sujet, nous avons suivi un ensemble de contraintes que nous énumérons ci-dessous :

- 8 classes
- 3 niveaux de hiérarchie
- 2 fonctions virtuelles différentes et utilisées à bon escient
- 2 surcharges d'opérateurs
- Diagrammme de classe UML complet
- pas de méthodes/fonctions de plus de 30 lignes (hors commentaires, lignes vides et assert)

# 2 Description de l'application développée

## 2.1 Fonctionnalités principales

Les fonctionnalités de notre jeu sont le saut de plateforme en plateforme, pour éviter de tomber dans le vide. Avec les flèches de votre clavier, ainsi que la barre espace, vous pourrez sauter pour atteindre les plateformes, et même courir pour faire des sauts plus longs.

## 2.2 Architecture globale

Nous avons modulé notre application pour qu'elle soit lisible par tous les utilisateurs de notre application. Celle-ci se scinde entre plusieurs codes, toutes les fonctionnalités ou principes sont compris dans un .hh et un .cc. Le module principal gère la boucle de jeu et coordonne les interactions entre les différentes entités du jeu.

Pour la version graphique, on dessine les arrières-plans et l'interface visuelle à l'aide de la librairie Raylib. A chaque instant, on transmet la position du joueur sous forme de coordonnées afin que la scène puisse se mettre à jour constamment. On a évidemment un module utilisateur, qui gère donc toutes les entrées clavier et souris pour les transmettre à la logique centrale.

Lorsque le joueur appuie sur une touche, ce module interagit avec le code principal pour effectuer l'action requise, comme sauter pour SPACE, ou bien aller vers la gauche pour LEFTKEY (flèche de gauche). Pour le stockage des ressources, un gestionnaire de fichiers charge des images.

Enfin, des tests unitaires et d'intégration veillent à la fiabilité du système, vérifiant le bon fonctionnement de chaque composant et l'harmonie entre eux.

# 3 Utilisation des contraintes

## 3.1 Hiérarchie des classes (3 niveaux)

La hiérarchie des classes facilite la structuration du code en instaurant des relations de parenté entre les différentes catégories, favorisant ainsi le transfert et la gestion des caractéristiques communes.

Dans notre code, nous avons établi une hiérarchie en trois étapes en prenant en compte les concepts suivants : Entity, Player et Score.

Effectivement, une entité est une catégorie qui comprend tout ce que nous souhaitons inclure. Il s'agit d'une création qui nous donne la possibilité de manipuler tout ce présent dans le jeu. La classe joueur incarne le joueur en tant qu'entité et hérite donc de la classe Entity.

Finalement, le score contrôle la performance du joueur et son apparition. Elle comprend des techniques pour l'ajout de points et la visualisation du score sur l'écran.

Elle interagit avec cette classe ainsi, en fonction des actions du joueur, elle va modifier le score, ce qui en fait une fonction de niveau 3 par rapport à la hiérarchie de base.

## 3.2 Fonctions virtuelles

Les fonctions virtuelles permettent de définir des comportements qui peuvent être redéfinis dans les classes dérivées.

Dans la classe Entity, on utilise la fonction virtuelle :

```
move(const Vector2 &delta)
```

Cette fonction est virtuelle, ce qui signifie que les autres classes peuvent redéfinir cette fonction pour gérer le déplacement de manière spécifique.

Par exemple, la classe Player pourrait gérer le mouvement en fonction des touches du clavier, tandis que Enemy pourrait avoir un mouvement automatique.

Chaque entité (joueur, ennemi) peut avoir sa propre logique pour gérer le mouvement tout en appelant cette méthode de manière générique.

Toujours dans la classe Entity, on a la fonction :

```
move(float x, float y)
```

Cette fonction nous permet de manipuler les coordonnées de toutes les entités de notre jeu. Et comme la fonction précédente, elle est virtuelle, ce qui signifie que les autres classes peuvent la redéfinir selon ce qui est le plus logique.

## 3.3 Surcharge d'opérateurs

La surcharge d'opérateurs permet de redéfinir le comportement des opérateurs, comme ce qui nous convient le mieux dans notre jeu.

Nous avons travaillé avec des surcharges sur les opérateurs d'assignation de valeur (=) et sur la comparaison des identités (==) lors de débugage de l'application.

Pour l'opérateur d'assignation, cela permet de copier les données d'un objet vers un autre. Il est utile pour ne pas à copier ou dupliquer des références internes ou des pointeurs, ce qui complexifierait le code, et ne compilerait pas.

Pour l'opérateur de comparaison d'entité, on l'utilise généralement pour comparer nos deux entités, comme un ennemi ou un joueur.

## 3.4 Conteneurs STL

Dans ce projet, nous utilisons principalement deux conteneurs STL pour gérer efficacement les objets du jeu. Le conteneur std::list¡Enemy¿ est utilisé pour stocker les ennemis, car il s'agit d'une liste doublement chaînée qui permet des ajouts et suppressions fréquents d'éléments de manière efficace en temps constant (O(1)).

Toutefois, l'accès direct aux éléments est moins rapide (O(n)), ce qui est acceptable dans ce contexte où les ennemis sont principalement gérés par itération. D'autre part, bien que non utilisé explicitement dans le code, des conteneurs comme std::map ou std::unordered_map seraient utiles pour associer des entités à des positions ou identifiants uniques.

Le std::map garde les éléments triés, tandis que std::unordered_map permet des recherches en temps constant (O(1)) grâce à une table de hachage. Ces conteneurs permettent une gestion rapide et efficace des entités, améliorant la performance et la lisibilité du code.

### 3.5 Commentaires du code

Les codes que nous avons fournis sont commentés afin que l'utilisateur sache ce qu'il lance sur sa machine. Il peut comprendre le code et ainsi savoir avec plus de précision ce qu'il peut faire.

## 4 Diagramme UML de l'application

### 4.1 Diagramme de classes

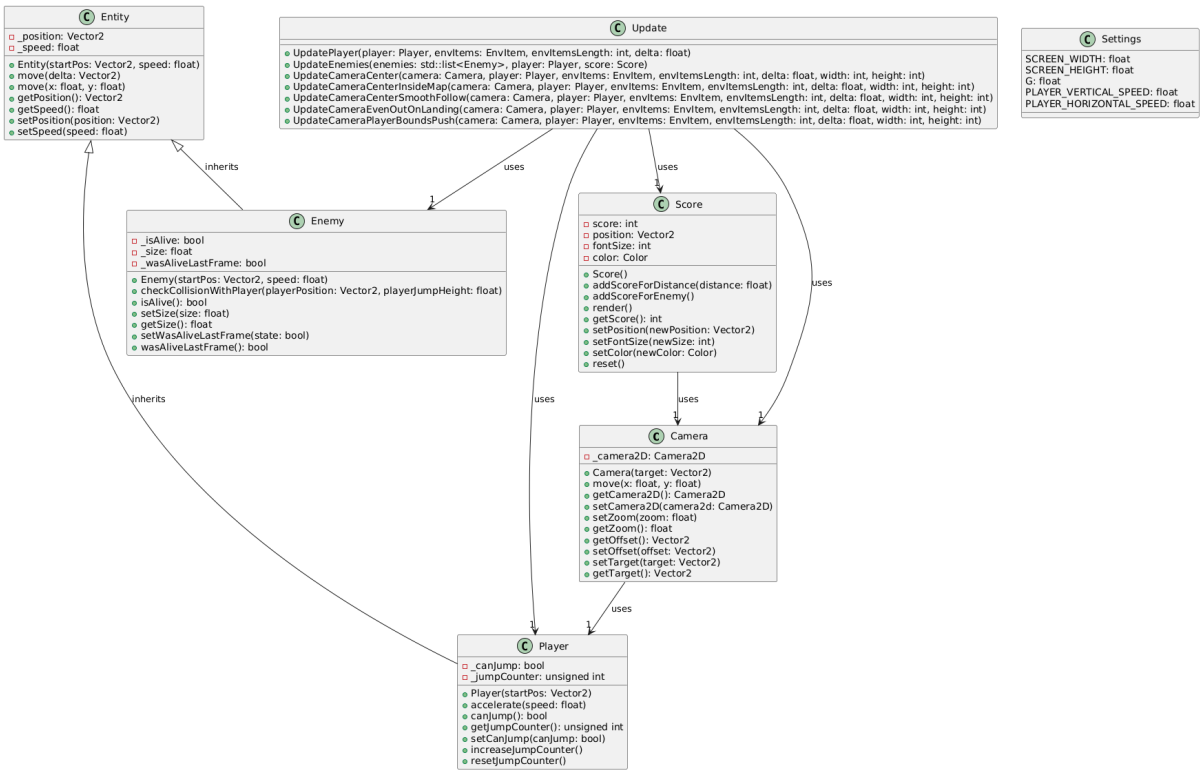



Figure 1: Le diagramme UML de notre application

### 4.2 Explication du diagramme

Le schéma UML illustre la configuration du projet, incluant les classes clés et leurs relations. Entity est la classe fondamentale qui regroupe des caractéristiques communes à toutes les entités du jeu, comme la position et la vitesse, ainsi que des techniques pour utiliser ces caractéristiques. Les classes Player et Enemy tirent leur inspiration de Entity et intègrent des comportements précis. Player supervise les mouvements et le saut du joueur, tandis qu'Enemy gère les relations interpersonnelles du joueur, incluant les collisions et les modifications d'état des adversaires.

La classe Camera administre la vue du jeu, facilitant le suivi du joueur et l'ajustement du zoom basé sur ses déplacements. Quant à la classe Score, elle est responsable de la gestion du score du joueur, ainsi que de son affichage à l'écran.

L'héritage et l'usage sont les principaux fondements des relations entre les différentes classes. Player et Enemy tirent leur inspiration de Entity, ce qui leur donne la possibilité d'adopter des comportements similaires tout en incorporant des caractéristiques distinctes. La classe Mise à jour coordonne toutes les

actualisations du jeu en employant Player, Enemy, Camera et Score pour contrôler les mouvements du joueur, des adversaires, la caméra et l'affichage du score.

Notamment, les fonctionnalités présentes dans Update facilitent la gestion des interactions entre ces entités, telles que le mouvement du joueur, la prise en charge des collisions avec les adversaires et l'actualisation de la caméra. En définitive, Entity sert de fondement pour les autres catégories tandis qu'Update garantit la coordination de toutes les composantes du jeu.

# 5 Installation des dépendances

Pour compiler et exécuter correctement ce projet, vous devez d'abord installer les dépendances suivantes :

1. **Raylib** (bibliothèque graphique) :
   - Sous Ubuntu/Debian, exécutez :
     `sudo apt-get install libraylib-dev`
   - Sous Windows, téléchargez la version appropriée sur
     `https://www.raylib.com/`
2. **CMake** (si nécessaire pour la configuration du projet) :
   - `sudo apt-get install cmake`
3. **(Optionnel) Autres bibliothèques utiles**, selon vos besoins.

Vous pouvez également suivre les instructions dans le README.

# 6 Compilation du code avec le Makefile

Le projet est fourni avec un fichier `Makefile` simplifiant la compilation.

1. Lancez la commande `make all` pour compiler l'ensemble du code :

   ```
   cmake --build .
   ```

2. L'exécutable sera alors généré dans le répertoire courant (ou dans un sous-répertoire `build/`)

Une fois la compilation terminée :

- **Exécuter l'application principale** :

  ```
  ./MarioShab
  ```

  (ou bien `./build/MarioShab` si l'exécutable se trouve dans `build/`).

# 7 Implémentation

## 7.1 Parties dont tu es fier

Nous avons accordé une attention particulière aux interactions entre le joueur et les autres éléments du jeu. Notre objectif est de garantir que chaque interaction, qu'il s'agisse d'une chute, d'un saut ou d'un déplacement vers la gauche ou la droite, se déroule de façon fluide et répétitive. Cela favorise une meilleure immersion dans le jeu, et nous a aidé à mieux appréhender notre code.

Nous sommes également fier d'avoir pu configurer une map énigmatique afin de proposer plusieurs niveau de difficulté sur la même partie. Nous avons nous même tester le jeu, afin de nous assurez qu'il est réalisable. (Si un certain niveau vous parait trop compliqué n'hésitez pas a regarder les commentaires sur le code.)

### 7.2 Défis rencontrés

Plusieurs niveaux de difficultés ont été rencontrés lors de la mise en œuvre de notre programme. En premier lieu, il s'est avéré difficile d'intégrer la bibliothèque Raylib et de comprendre ses caractéristiques. Un autre défi a été la création de tests unitaires adaptés à une application graphique. Effectivement, au cours de nos cours, nous n'avons observé que des exemples de tests qui ne conviennent pas à des projets utilisant une interface visuelle, comme c'était le cas lors du TP6. Il nous a donc fallu faire preuve d'innovation et développer nous-mêmes des tests unitaires pertinents pour ce genre de code. Les collisions au dessus et sur les cotés du joueur prennaient trop de temps, et en vue de respecter la deadline, ont été abandonnées.

## 8 Conclusion

En considérant l'ensemble de l'élaboration de ce projet, nous pouvons conclure que travailler en autonomie a été une excellente expérience pour découvrir et approfondir le langage C++. Ce processus nous a également permis de nous confronter à des tâches variées, comme la rédaction d'un `README`, la mise en place et la maintenance d'un dépôt Git, ou encore la gestion du versionnement au sein d'un groupe. Notre idée de créer un jeu de plateforme illustre parfaitement la manière dont il est possible de concilier imagination et compétences de développement, afin de réaliser des projets qui nous font progresser sur les plans techniques et créatifs.

### 8.1 Améliorations possibles

Même si l'application est déjà en fonctionnement, il est possible d'envisager plusieurs pistes d'amélioration pour l'avenir. De nouvelles mécaniques de jeu pourraient être ajoutées, comme des ennemis plus diversifiés ou des objets à collecter, ou encore des énigmes à résoudre pour enrichir l'expérience globale.

Grâce à l'amélioration des performances, que ce soit en diminuant l'empreinte mémoire, en améliorant la fluidité de l'animation ou en optimisant les calculs de physique, l'exécution serait également plus réactive. D'autres options comprennent la mise en place d'une extension multiplate-forme afin de simplifier le déploiement sur divers systèmes d'exploitation, ou encore la création d'une interface utilisateur plus sophistiquée, comprenant un système de menus, un tableau des scores ou des écrans de tutoriel des nouveaux joueurs.

Enfin, la conception d'une fonctionnalité de sauvegarde et chargement de partie constituerait un ajout précieux pour que le joueur puisse reprendre sa progression là où il l'a laissée. Nous avions commencé aussi a codé un éditeur de niveau qui permettrait au joueur de créer son propre niveau. Celui-ci se matérialisait par une grille dont il fallait sélectionner les cases qui aillaient être représenté par des plateformes. Cependant plusieurs bug ont été rencontré, une limite de bloc à poser était imposée, ce qui nous a contrait à abandonner cette idée.

### 8.2 Retour d'expérience

Sur le plan personnel, ce projet a été à la fois motivant et instructif. Il est indéniable que la découverte et la mise en œuvre d'un moteur physique ont été la partie la plus enrichissante, car cela m'a permis de renforcer mes compétences en C++ tout en affrontant un défi technique. La gestion de projet en équipe m'a également plu, car elle nous a incités à structurer notre travail et à faire un suivi régulier via Git.

D'autre part, la création de tests unitaires adaptés à une application graphique s'est avérée plus complexe, car ce sujet était peu traité lors des cours. Nous avons donc dû chercher des solutions par nous-mêmes, ce qui s'est avéré instructif et nous a permis de mieux comprendre l'importance de la couverture de tests, même pour des projets orientés vers une interface graphique.