



UNIVERSITE :

FACULTE DES SCIENCES SEMLALIA MARRAKECH (FSSM)

TITRE DU PROJET :

DEVELOPPEMENT D'UN JEU 2D AVEC RAYLIB

REALISE PAR :

- YASSER EL FALLAH
- YOUSSEF CHOUABI

ANNEE UNIVERSITAIRE :

2024/2025

Rapport de Projet : Développement d'un Jeu 2D avec Raylib

1. Introduction

Ce projet consiste en la création d'un jeu 2D de type "runner" développé en C++ avec la bibliothèque Raylib. Le jeu met en scène un personnage (le joueur) qui doit collecter des pièces tout en évitant des ennemis pour atteindre un objectif final. Le jeu intègre plusieurs états (menu principal, jeu, écrans de victoire/défaite) et des mécaniques de base comme le saut, les collisions et une caméra dynamique.

L'objectif de ce rapport est de présenter la conception du jeu, les choix techniques, et les principes de programmation orientée objet (POO) appliqués. Le rapport décrit également l'interface et les fonctionnalités.

Ce projet illustre plusieurs concepts clés :

- Gestion des entrées utilisateur
- Physique de mouvement et gravité
- Système de collision
- Gestion des états du jeu
- Rendu graphique optimisé

2. Spécification des Besoins

2.1 Besoins Fonctionnels

Besoins Principaux :

1. Mouvement du Joueur :

- Déplacement horizontal fluide
- Mécanique de saut avec gravité
- Détection du sol

2. Système d'Ennemis :

- Comportement de déplacement automatique
- Réapparition après sortie d'écran
- Gestion des collisions avec le joueur

3. Environnement de Jeu :

- Génération de plateformes
- Éléments décoratifs (arbres, nuages)

- Zone de victoire

4. Gestion des États :

- Écran d'accueil ○ État de jeu principal
- Écrans de fin (victoire/défaite)

2.2 Besoins Non-Fonctionnels

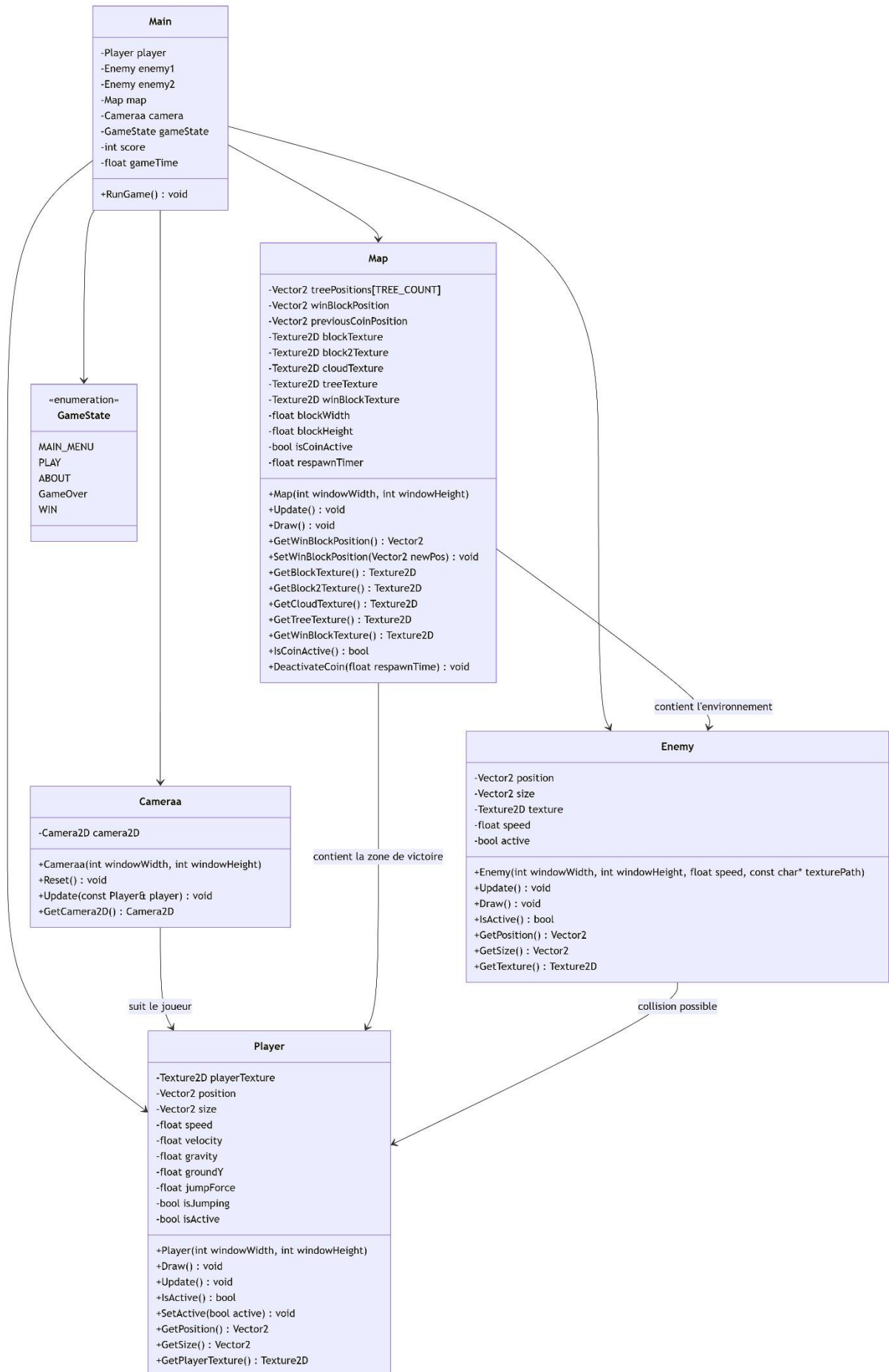
Performance :

- Maintien d'un framerate stable à 60 FPS
- Chargement optimisé des ressources
- Gestion mémoire efficace
- Contrôles intuitifs
- Feedback visuel clair
- Transitions fluides entre les états
- Code modulaire et documenté
- Architecture extensible
- Respect des principes SOLID

Expérience Utilisateur :

Maintenabilité :

3. représentation des classes avec un diagramme UML :



Explications :

1. Classes Principales :

- Player : Gère le personnage jouable avec physique de saut et déplacement
- Enemy : Gère les ennemis avec déplacement automatique et réapparition
- Map : Contient tout l'environnement (plateformes, arbres, pièce de victoire)
- Cameraa : Caméra dynamique qui suit le joueur
- GameState : Enumération des états du jeu (menu, jeu, victoire, etc.)

2. Relations :

- **Composition** : La classe Main contient toutes les autres instances
- **Dépendances** :
 - Cameraa dépend de Player pour le suivi
 - Map fournit l'environnement pour Player et Enemy
 - Collisions entre Player et Enemy/Map

3. Attributs/Méthodes :

- Tous les membres privés sont listés avec -
- Méthodes publiques avec +
- Types Raylib (Vector2, Texture2D) explicités

4. Particularités :

- La caméra est nommée Cameraa (attention à la faute de frappe dans le code)
- Système de pièces/score géré dans Map et Main
- États du jeu clairement séparés via l'enum GameState

Ce diagramme reflète l'architecture actuelle de votre jeu runner 2D et montre comment les différentes classes interagissent entre elles.

4. Structure des Classes

4.1 Classe Player

- **Attributs :**
 - position : Coordonnées du joueur.
 - size : Taille du joueur.
 - speed : Vitesse de déplacement horizontal.
 - velocity : Vitesse verticale (pour le saut).
 - gravity : Force de gravité appliquée au joueur.
 - isJumping : État du saut.
 - isActive : État d'activité du joueur.
- **Méthodes :**
 - Update() : Met à jour la position et l'état du joueur.
 - Draw() : Dessine le joueur à l'écran.
 - IsActive() : Retourne l'état d'activité.
 - SetActive(bool) : Modifie l'état d'activité.

4.2 Classe Enemy

- **Attributs :**
 - position : Coordonnées de l'ennemi.
 - size : Taille de l'ennemi.
 - speed : Vitesse de déplacement.
 - active : État d'activité de l'ennemi.
- **Méthodes :**
 - Update() : Met à jour la position de l'ennemi.
 - Draw() : Dessine l'ennemi à l'écran.

4.3 Classe Map

- **Attributs :**
 - treePositions : Positions des arbres dans le niveau.
 - winBlockPosition : Position du bloc de victoire.
 - Textures pour les blocs, arbres, nuages, etc.
- **Méthodes :**

- Update() : Met à jour les éléments du niveau (ex : déplacement des arbres).
- Draw() : Dessine le niveau à l'écran.

4.4 Classe Camera

- **Attributs** : ○ camera2D : Objet caméra de Raylib.
- **Méthodes** :
 - Update(const Player&) : Met à jour la position de la caméra pour suivre le joueur.
 - GetCamera2D() : Retourne l'objet caméra.

4.5 Classe Principale (main)

- Gère la boucle de jeu, les états, et les interactions entre les objets.

5. Principes de la POO Appliqués

5.1 Encapsulation

- Les attributs des classes sont privés et accessibles via des méthodes (ex : GetPosition()).
- Exemple : La classe Player encapsule la logique de saut et de déplacement.

5.2 Composition

- La classe main compose les objets Player, Enemy, Map, et Camera.
- Exemple : La caméra est un composant distinct qui suit le joueur.

5.4 Polymorphisme

- Potentiel pour des méthodes communes (ex : Update() et Draw()), mais non implémenté dans cette version.

6. Environnement de Travail

Outil	Rôle
Raylib	Bibliothèque graphique pour le rendu et les entrées.
GCC/Clang	Compilation du code C++.
Visual Studio Code	Éditeur de code avec extensions C++.

7. Interface de Jeu et Descriptions

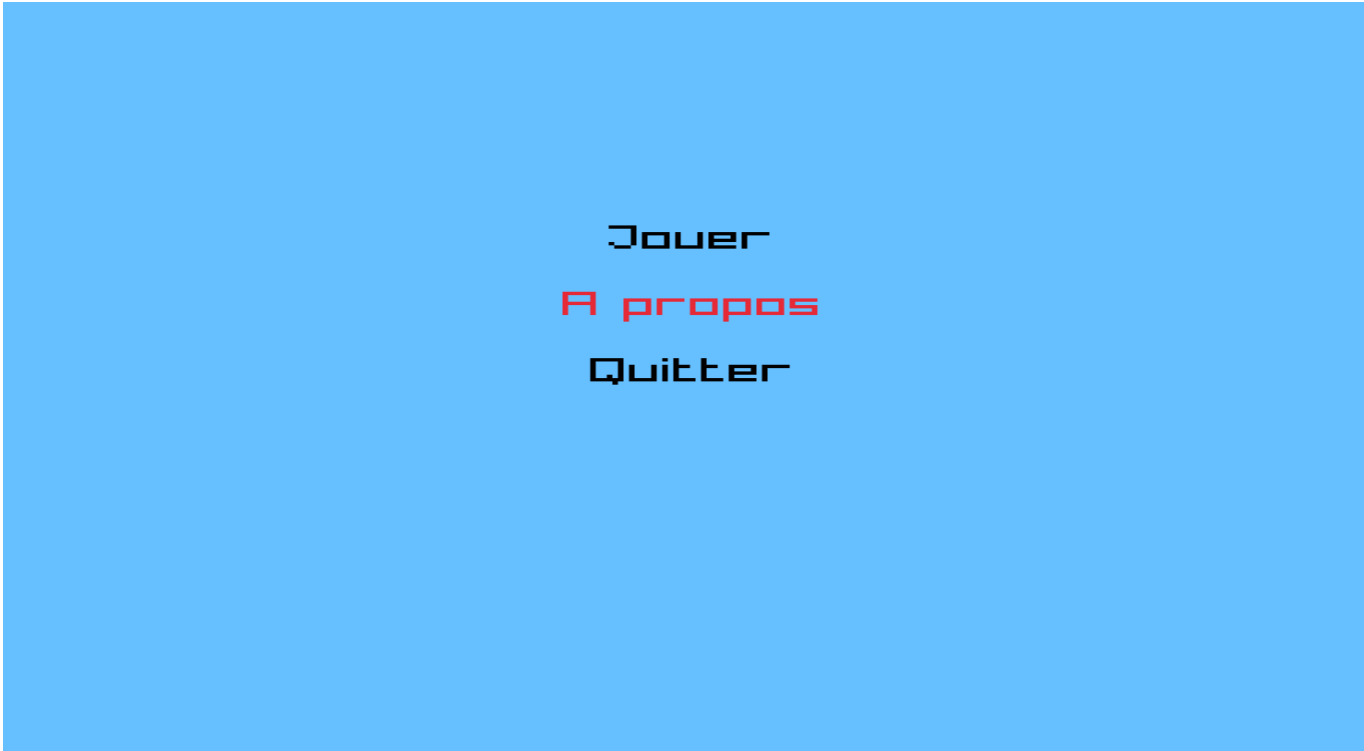
Figure 1 : Menu Principal

- **Description :** Affiche les options "Jouer", "À propos ", "quiter".



Figure 2 : A propos

- **Description** : lorsque vous appuyez sur "A propos ", des instructions apparaissent pour vous aider a jouer .



Jouer
A propos
Quitter



Controles:
A/D - Deplacement
ESPACE - Saut

Figure 3 : quitter

- **Description** : lorsque vous appuyez sur quitter , vous allez quitter le jeu



Figure 4 : jouer

- **Description** : lorsque vous appuyez sur jouer , le jeu commence



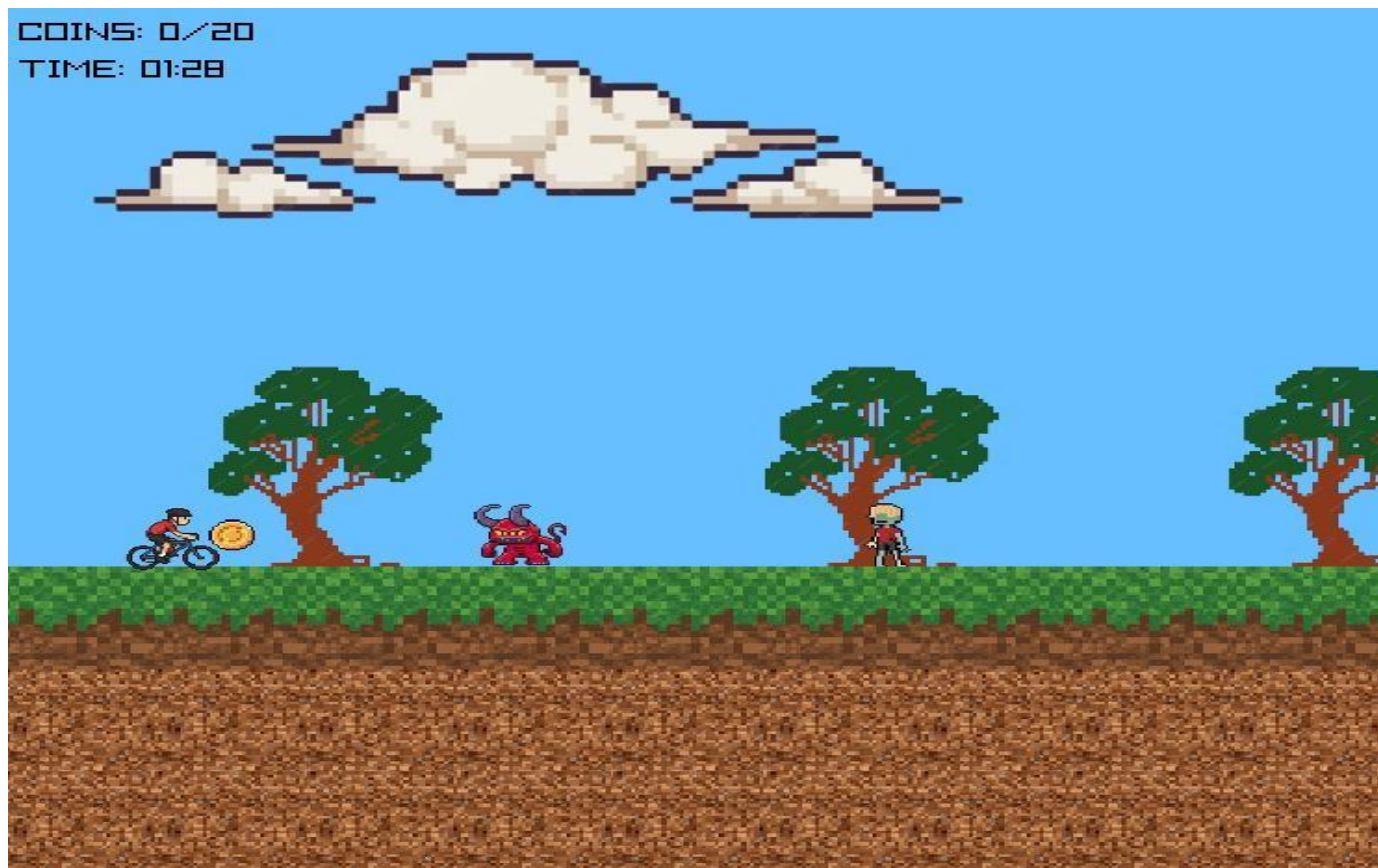


Figure 5 : Écran de Victoire

- **Description** : Affiche "YOU WIN!" lorsque le joueur collecte les coins avant l'épuisement du temps.



Figure 6 : Écran de Défaite

- **Description** : dans le cas de défaite il ya deux cas :

1ere cas : c'est le temps est epuise :



2eme cas : c'est l'ai touche l'ennmis :



7. Conclusion

Ce projet de développement d'un jeu 2D de type "runner" avec Raylib a permis de mettre en pratique des concepts fondamentaux de la programmation orientée objet (POO) ainsi que des mécaniques de jeu essentielles. Grâce à une architecture modulaire et bien structurée, les différentes fonctionnalités, telles que le mouvement du joueur, la gestion des ennemis, les collisions et les états du jeu, ont pu être implémentées de manière efficace et maintenable.

Les principes de la POO, comme l'encapsulation, la composition et le polymorphisme, ont été appliqués pour garantir un code clair et extensible. L'utilisation de Raylib a facilité la gestion du rendu graphique et des entrées utilisateur, tout en maintenant des performances optimales avec un framerate stable à 60 FPS.

L'interface intuitive, comprenant un menu principal, des écrans de victoire et de défaite, ainsi que des éléments visuels clairs, contribue à une expérience utilisateur agréable. Les défis rencontrés, tels que la gestion des collisions et le suivi de la caméra, ont été surmontés grâce à une approche méthodique et itérative.

En somme, ce projet démontre non seulement la maîtrise des outils et des concepts techniques, mais aussi la capacité à concevoir un jeu complet et fonctionnel. Il ouvre également des perspectives d'amélioration, comme l'ajout de niveaux supplémentaires, de nouvelles mécaniques de jeu ou l'optimisation des assets graphiques. Ce travail constitue une base solide pour des développements futurs plus ambitieux.