PROJET DE FIN DE MODULE
Programmation orientée objet en C++

APPLICATION DE JEU DE COURSE À VÉLO

Université : CADI AYYAD

Faculté : FACULTÉ DES SCIENCES SEMLALIA - MARRAKECH

Département : Informatique

Année universitaire : 2024 – 2025

Encadrante : Mme R.Hannane

Binôme :

• Étudiant 1 : AMINE RIHAN – 2130072 – INF04 – grp 03

• Étudiant 2 : ADHAM EL WARARI - 2130071 - INF04 – grp 01

Date de remise : 10/05/2025

Table des matières

Table des matières

1. Introduction

Dans le cadre du module de **programmation orientée objet (POO)** dispensé au cours de l'année universitaire 2024–2025, il nous a été proposé de concevoir et de développer une application logicielle complète en langage **C++**, en mettant en pratique les concepts fondamentaux du paradigme orienté objet. Pour répondre à cet objectif pédagogique, nous avons choisi de réaliser un **jeu de course à vélo** basé sur une interface graphique interactive et un système de logique simple, mais robuste.

Ce projet consiste en un jeu vidéo mono-joueur dans lequel le joueur incarne un cycliste engagé dans une course contre la montre. L'objectif principal du joueur est de parcourir une piste en évitant des obstacles fixes (représentés par des murs) tout . Si le joueur entre en collision avec un mur, la partie est immédiatement perdue. Dans ce cas, une nouvelle interface de menu s'affiche automatiquement avec un bouton "Rejouer", permettant au joueur de relancer la partie sans devoir redémarrer entièrement l'application.

L'interface du jeu est conçue de manière à être **simple, ergonomique et interactive**. Dès le lancement de l'application, un **menu principal** s'affiche, comportant plusieurs options accessibles via la **souris** : "Jouer", "À propos", et "Quitter". L'option "À propos" permet d'informer le joueur sur le fonctionnement et les règles du jeu. Le menu est également réaffiché à la fin de chaque partie, ce qui favorise une navigation fluide et intuitive. L'affichage d'un **score final** et du **temps écoulé** après chaque session permet au joueur d'évaluer ses performances, ajoutant ainsi une dimension de challenge personnel.

Le jeu exploite la **bibliothèque SFML (Simple and Fast Multimedia Library)**, qui permet une gestion efficace de la fenêtre graphique, des événements utilisateurs, des formes géométriques, du texte, ainsi que de la musique. Grâce à SFML, nous avons pu intégrer une **ambiance sonore** qui accompagne l'utilisateur tout au long de l'expérience de jeu, renforçant ainsi l'aspect immersif du projet. Le vélo, les murs, le texte, le chronomètre, ainsi que tous les éléments visuels ont été gérés via les classes fournies par la SFML.

Le développement a été réalisé à l'aide de l'environnement **Code::Blocks**, associé au compilateur **MinGW g++**. L'architecture du projet a été pensée de manière modulaire et respecte les **principes fondamentaux de la programmation orientée objet** : encapsulation des données, responsabilité unique, séparation des préoccupations, abstraction, réutilisabilité du code, et facilité d'extension.

Ce projet constitue une synthèse des notions vues durant les semestres précédents. Il nous a permis non seulement de renforcer nos compétences en C++ et en POO, mais également de comprendre les enjeux liés à la conception d'une application interactive graphique. Le choix de créer un jeu, bien que relativement simple, a présenté de nombreux défis techniques, notamment en matière de gestion d'événements, de détection de collisions, de traitement du temps réel et de synchronisation des éléments graphiques et sonores.

Ce rapport présente dans les sections suivantes l'ensemble des étapes du développement, depuis la spécification des besoins jusqu'à l'implémentation finale. Il met également en lumière la structure des classes, l'architecture logicielle choisie, ainsi que l'environnement de développement utilisé. Des

captures d'écran viendront illustrer les différents aspects de l'interface utilisateur. Enfin, nous conclurons par un bilan sur les limites du projet actuel et les perspectives d'amélioration possibles.

2. Spécification des besoins

Le développement du jeu repose sur l'analyse précise des besoins fonctionnels et non fonctionnels afin de garantir une expérience utilisateur cohérente et conforme aux objectifs pédagogiques du projet. Les fonctionnalités ci-dessous ont été définies et implémentées pour assurer la fluidité du jeu, l'interactivité de l'interface, et la robustesse de l'application dans son ensemble.

2.1. Fonctionnalités fonctionnelles

Le jeu a été conçu avec une série de fonctionnalités essentielles visant à assurer un déroulement intuitif et attractif. Dès le lancement de l'application, l'utilisateur est accueilli par un **menu graphique interactif** contenant trois options principales :

- **Jouer** : permet de lancer immédiatement une partie.
- **À propos** : fournit des informations sur le projet.
- **Quitter** : ferme l'application.

La navigation dans le menu s'effectue entièrement à la **souris**, permettant une interaction rapide, fluide et intuitive. Chaque bouton est cliquable et déclenche une action associée sans délai perceptible.

Lorsqu'une partie débute, l'écran de jeu s'affiche et présente les éléments suivants :

- Le **vélo**, représenté par une forme graphique simple, contrôlé par les **touches fléchées** du clavier pour se déplacer dans différentes directions.
- Des **murs fixes** apparaissent sur le parcours et représentent des obstacles à éviter.

La logique de jeu repose sur une condition de défaite :

- Une **collision** entre le vélo et un mur entraîne une fin de partie immédiate.

Après la defaite , un écran de fin de partie est généré, affichant :

- Le **score final** du joueur.
- Le **temps total** passé dans la partie.

À l'issue de la défaite, le **menu principal est automatiquement réaffiché**, avec une modification : le bouton **"Jouer"** est remplacé par le bouton **"Rejouer"**, permettant à l'utilisateur de relancer rapidement une nouvelle tentative sans quitter le jeu.

Enfin, une **ambiance sonore** (musique de fond) est jouée en boucle durant toute la durée du jeu, depuis le menu principal jusqu'à la fin de la partie, renforçant l'immersion dans l'univers du jeu.

2.2. Fonctionnalités non fonctionnelles

Outre les fonctionnalités attendues par le joueur, plusieurs exigences techniques et ergonomiques ont été respectées lors du développement :

- L'application offre une **interface fluide** avec un **taux de rafraîchissement stable** (≥ 60 images par seconde), garantissant une expérience sans latence ni ralentissements.
- Le projet repose exclusivement sur l'utilisation de la **bibliothèque SFML**, qui gère l'ensemble des aspects graphiques, sonores et événementiels.
- Le code source est structuré en suivant strictement les principes de la **programmation orientée objet** : séparation des responsabilités, encapsulation, modularité.
- Le système d'interaction combine deux types de navigation :
    - **Souris** pour les menus et boutons.
    - **Clavier** pour les déplacements en jeu.
- L'affichage des **textes**, **boutons**, **éléments graphiques** (vélo, murs, chronomètre) est **responsive** et adapté aux dimensions de la fenêtre du jeu, ce qui garantit une bonne lisibilité quel que soit le poste utilisé.

L'ensemble de ces fonctionnalités a été implémenté avec soin afin d'offrir une expérience utilisateur satisfaisante, respectant les standards d'un jeu léger, intuitif et fonctionnel, tout en illustrant concrètement l'application de la programmation orientée objet en contexte réel.

3. Outils et gestion de projet

Cette section présente les outils utilisés tout au long du développement de l'application de jeu de course à vélo. Ces outils ont permis de garantir une gestion fluide du projet, ainsi qu'une collaboration efficace entre les membres du binôme.

3.1. Outils et technologies utilisés

Les principaux outils et technologies utilisés pour ce projet sont les suivants :

- **IDE : Code::Blocks 20.03**
  Code::Blocks est un environnement de développement intégré (IDE) qui offre une interface complète pour le développement en C++. Il permet une gestion avancée du projet, des fonctionnalités de débogage et d'édition de code, et simplifie la création de nouveaux fichiers source et leur compilation.
- **Compilateur : MinGW g++**
  MinGW est un environnement de compilation pour Windows qui permet d'utiliser le compilateur g++ basé sur GCC (GNU Compiler Collection). Cela permet de compiler le code C++ en un fichier exécutable fonctionnel.

- **Système d'exploitation : Windows 11**
  Le développement s'est effectué sur Windows 11, un système d'exploitation fiable et largement utilisé, compatible avec les outils choisis pour ce projet.
- **Format d'image : Primitives SFML**
  Le jeu utilise des primitives graphiques SFML, telles que des rectangles, pour représenter les objets du jeu, tels que le vélo et les murs. Cette approche permet de maintenir une gestion efficace et simple des éléments visuels sans avoir à charger des images externes complexes.
- **Son : .ogg géré via sf::Music**
  Les effets sonores et la musique de fond sont gérés à l'aide de la classe sf::Music de SFML. Les fichiers sonores sont au format .ogg et sont joués en boucle pendant la durée de la partie, contribuant à l'ambiance sonore du jeu.

3.2. Structure du projet

Le projet est structuré de manière simple et fonctionnelle, avec tous les fichiers regroupés à la racine du dossier principal, situé sur le Bureau. Voici les principaux éléments qui composent l'organisation du projet :

- **Fichiers sources principaux** : main.cpp, BikeGameSFML.cpp et BikeGameSFML.hpp forment le cœur de l'application. Ils contiennent la logique du jeu, les entités (vélo, arbres, obstacles, etc.) et les interactions graphiques avec la bibliothèque **SFML**.
- **/assets** : Dossier contenant toutes les ressources multimédia du jeu, notamment les images (textures), les sons, et la police utilisée pour l'interface graphique.
- **/bin** : Ce répertoire contient l'exécutable généré après la compilation du projet.
- **/obj** : Répertoire intermédiaire qui stocke les fichiers objets (.o) compilés à partir du code source.
- **Fichiers de projet Code::Blocks** :
  - projetCourseBike.cbp : fichier principal du projet.
  - projetCourseBike.depend et projetCourseBike.layout : fichiers automatiques gérant les dépendances et l'interface de développement dans Code::Blocks.

3.3. Gestion de projet et communication : Git

Dans un projet de binôme, la gestion de version est essentielle pour une collaboration efficace. Git a été utilisé pour suivre l'historique des modifications et pour faciliter les échanges entre les deux membres.

- **Plateforme : GitHub**
  Le projet a été hébergé sur GitHub, permettant de centraliser le code source, suivre les différentes versions du projet, et collaborer efficacement. GitHub permet également de

gérer les demandes de fusion (pull requests), ce qui facilite la gestion des contributions de chaque membre.

- **Flux de travail Git**
  Les deux membres ont utilisé un système de branches pour développer différentes fonctionnalités de manière indépendante, avant de les fusionner avec la branche principale (main). Cela a permis de maintenir une version stable et à jour du projet tout au long de son développement.

4. Conception orientée objet

L'architecture du jeu a été conçue en respectant les principes de la **programmation orientée objet (POO)**, un paradigme qui permet d'organiser le code en objets représentant les différents composants du jeu (vélo, obstacles, interface utilisateur, etc.). Cette approche assure une **gestion claire, modulaire et évolutive** du projet, facilitant ainsi sa maintenance et l'ajout de nouvelles fonctionnalités.

4.1. Structure des fichiers

Le projet est structuré autour de plusieurs fichiers sources qui assurent une bonne séparation des responsabilités :

- **main.cpp** : point d'entrée de l'application. Il instancie la classe Game, initialise le jeu, puis lance la boucle principale.
- **BikeGameSFML.hpp** : contient la déclaration de l'ensemble des classes utilisées dans le jeu, y compris la classe principale Game ainsi que les classes secondaires (Bike, Wall, Token, Tree, Button).
- **BikeGameSFML.cpp** : regroupe l'implémentation des classes et la logique du jeu : gestion des menus, rendu graphique, mises à jour, collisions, etc.

4.2. Classe principale : Game

La classe Game est le **cœur du projet**. Elle gère l'état global du jeu, le rendu, les interactions utilisateur, la navigation entre les menus, et la logique de gameplay.

Principaux attributs :

- **Fenêtre graphique (sf::RenderWindow)** : permet d'afficher les éléments visuels (vélo, obstacles, arrière-plan, texte).
- **Chronomètre (sf::Clock + sf::Time)** : mesure le **temps écoulé** depuis le début de la partie
- **État du jeu (GameState currentState)** : permet de savoir si le joueur est dans le menu, en jeu, dans l'écran "à propos", ou sur l'écran de fin.
- **Score (int score)** : score du joueur, incrémenté à chaque mise à jour du jeu.
- **Objet vélo (Bike* bike)** : représente le joueur. Il se déplace horizontalement et évite les obstacles.

- **Obstacles (std::vector<Wall> walls)** : murs que le joueur doit éviter.
- **Objets à collecter (Token* token)** : bonus qui apparaissent aléatoirement sur la route.
- **Décor (std::vector<Tree> leftTrees, rightTrees)** : arbres défilant de part et d'autre de la route pour simuler le mouvement.
- **Musique de fond (sf::Music)** : ajoute une ambiance sonore immersive.
- **Boutons (Button*)** : composants interactifs dans les menus ("Jouer", "À propos", "Quitter", etc.).

Méthodes principales :

- **run()** : boucle principale du jeu. Elle gère l'enchaînement des événements, la mise à jour du jeu et le rendu à chaque frame.
- **handleEvents()** : capture les événements clavier et souris pour réagir aux interactions du joueur (déplacements, clics).
- **update()** : met à jour l'état du jeu (déplacement des éléments, collisions, score, temps écoulé).
- **render()** : dessine tous les éléments à l'écran selon l'état du jeu (menu, partie en cours, game over, etc.).
- **resetGame()** : réinitialise les éléments du jeu lorsqu'un joueur redémarre une partie.
- **renderMenu() / renderAbout() / renderGameOver()** : affichent respectivement le menu principal, la page "à propos" et l'écran de fin.
- **checkCollision()** : méthode utilitaire permettant de détecter les collisions entre le vélo et les obstacles ou les bonus.

4.3. Autres classes clés

- **Bike** : représente le vélo contrôlé par le joueur. Gère les déplacements via le clavier.
- **Wall** : représente les obstacles fixes sur la route. Leur position est réinitialisée lorsqu'ils sortent de l'écran.
- **Token** : objet collectable. Lorsqu'il est attrapé par le joueur, il disparaît temporairement avant de réapparaître.
- **Tree** : éléments de décor animés pour donner l'illusion de mouvement.
- **Button** : éléments d'interface utilisateur pour interagir avec le menu.

La conception du jeu respecte les bonnes pratiques de la programmation orientée objet : chaque classe a une **responsabilité unique**, l'interaction entre les composants est bien définie, et la structure modulaire du code rend le projet **lisible, maintenable et extensible**

5. Paradigme Orienté Objet appliqué

Le projet de jeu de course à vélo s'appuie pleinement sur les **principes fondamentaux de la programmation orientée objet (POO)**. Cette approche permet d'assurer une architecture **modulaire**, **maintenable**, et **réutilisable**, tout en garantissant une expérience utilisateur fluide. Les concepts clés de la POO — **encapsulation**, **abstraction**, **modélisation par états**, **responsabilité unique**, et **réutilisabilité** — sont rigoureusement appliqués tout au long du développement du jeu.

5.1. Encapsulation

L'encapsulation consiste à **protéger les données internes d'un objet** en les rendant inaccessibles depuis l'extérieur, sauf via des méthodes contrôlées. Elle est mise en œuvre dans ce projet de plusieurs manières :

- Les attributs sensibles de la classe Game (comme la fenêtre de jeu, le score, le chrono, ou les objets graphiques) sont déclarés en **privé**, empêchant tout accès direct non autorisé.
- Les actions sur ces données passent par des **méthodes publiques**, comme update() ou handleEvents(), ce qui garantit une **cohérence de l'état interne** du jeu.
- Cette encapsulation renforce la **sécurité du code**, tout en rendant le système plus robuste face aux erreurs ou modifications imprévues.

5.2. Abstraction

L'abstraction permet de **masquer la complexité** d'un système en exposant uniquement une interface claire et simple à utiliser.

- Le joueur interagit uniquement avec **l'interface graphique** du jeu, sans se soucier de la logique interne (détection des collisions, génération des obstacles, calculs de positions…).
- Des méthodes comme handleCollision(), render(), ou update() traitent des aspects complexes du gameplay **sans exposer leur implémentation**.
- Cette abstraction facilite non seulement l'utilisation du jeu par l'utilisateur, mais aussi **la lecture et la maintenance du code** par les développeurs.

5.3. Modélisation par états

Le jeu est structuré autour d'un système de **gestion des états**, conforme au modèle de **machine à états finis**. Cela garantit un comportement clair et prévisible.

Les états sont définis via un énuméré GameState, et chaque état est géré indépendamment :

- **MENU** : état d'accueil du jeu, avec navigation via des boutons ("Jouer", "À propos", "Quitter").

- **PLAYING** : phase active où le joueur contrôle le vélo, évite les murs, et collecte des bonus.
- **ABOUT** : état d'information, expliquant les règles et les commandes du jeu.
- **GAME_OVER** : écran de fin de partie affichant le score final, avec possibilité de rejouer ou quitter.

Cette séparation des états permet de **centraliser la logique de chaque phase**, rendant les transitions plus lisibles et évitant les interactions non souhaitées entre les différents composants.

5.4. Responsabilité unique

Chaque classe et chaque méthode respecte le principe de **responsabilité unique (Single Responsibility Principle)** :

- Les classes comme Bike, Wall, Token, Tree, ou Button sont conçues pour représenter **un seul type d'objet** avec une logique dédiée.
- La méthode handleCollision() se concentre exclusivement sur la gestion des collisions.
- Le rendu graphique est géré uniquement dans la méthode render(), et les entrées utilisateur dans handleEvents().

Ce découpage précis du code facilite la **compréhension**, la **correction** d'erreurs, et l'**ajout de fonctionnalités** sans effet secondaire inattendu.

5.5. Réutilisabilité

De nombreux composants du projet sont conçus pour être **réutilisables** dans d'autres jeux ou projets SFML :

- Les classes Bike, Wall, et Token peuvent facilement être adaptées à d'autres types de jeux en modifiant leur comportement.
- Les méthodes update(), resetGame(), ou checkCollision() sont génériques et pourraient être transposées dans d'autres contextes ludiques ou interactifs.
- La bibliothèque **SFML**, utilisée pour l'audio, le graphisme et les événements, facilite cette réutilisation à travers une API unifiée et portable.

L'application rigoureuse des **principes de la programmation orientée objet** a permis de concevoir une base de code **robuste, évolutive et claire**. Grâce à une structure bien pensée, le projet peut facilement être **étendu, modifié ou amélioré**, tout en assurant une **expérience utilisateur intuitive**. La POO n'est pas seulement un choix technique ici, mais une **véritable stratégie de conception** qui sert autant le développeur que le joueur.

6. Bibliothèques utilisées

Le développement du jeu de course à vélo s'appuie sur plusieurs bibliothèques essentielles, chacune remplissant un rôle spécifique pour assurer la fluidité, la performance et l'interactivité de l'application. Deux bibliothèques principales ont été utilisées : **SFML** pour la gestion multimédia et **la bibliothèque standard C++ (STL)** pour la manipulation des données et des opérations générales.

6.1. SFML (Simple and Fast Multimedia Library) – Version 2.5.1

SFML constitue la pierre angulaire du moteur graphique et sonore du jeu. Conçue pour le développement de jeux 2D, cette bibliothèque propose une interface simple et efficace pour gérer les éléments visuels, l'audio et les événements utilisateurs.

1. Gestion graphique

- Grâce à la classe sf::RenderWindow, SFML permet de créer une fenêtre de rendu dans laquelle les éléments du jeu sont affichés en temps réel.
- Les entités du jeu, comme le vélo et les murs, sont représentées à l'aide de formes simples (sf::RectangleShape, sf::CircleShape) dessinées à chaque frame.
- Cette approche offre un excellent compromis entre simplicité, lisibilité du code et performance, adaptée à un jeu 2D léger.

2. Son et musique

- L'ambiance sonore est assurée par la classe sf::Music, qui permet de lire en boucle des pistes audio au format .ogg.
- Des effets sonores ponctuels (comme les collisions) sont gérés par la classe sf::Sound, ajoutant du dynamisme et renforçant l'immersion du joueur.
- L'utilisation intégrée de ces classes évite le recours à des bibliothèques audio externes plus complexes.

3. Gestion des événements

- SFML facilite la détection des actions de l'utilisateur via la classe sf::Event, capable d'interpréter les entrées clavier et souris.
- Les actions principales (navigation dans les menus, contrôle du vélo) sont déclenchées en fonction des événements capturés, rendant le jeu interactif et réactif.
- Ce système d'événements centralisé simplifie le traitement des interactions dans la boucle de jeu.

6.2. STL (Standard Template Library) – Bibliothèque standard C++

La STL complète l'utilisation de SFML en fournissant des structures de données et des outils performants pour gérer les aspects non graphiques du jeu.

1. iostream

- Utilisée pour l'affichage d'informations dans la console, notamment à des fins de **débogage** ou pour afficher des **informations en fin de partie** (score, temps écoulé).
- Bien que limitée dans l'interface finale, cette bibliothèque reste utile durant le développement pour la vérification de l'état du jeu.

2. vector

- Le conteneur std::vector est central pour la **gestion dynamique des obstacles** (murs), stockés sous forme de sf::RectangleShape.
- Il permet d'ajouter ou supprimer facilement des éléments pendant l'exécution, tout en évitant la gestion manuelle de la mémoire.
- Ce choix garantit une **flexibilité maximale** dans l'évolution du gameplay.

3. string

- La classe std::string est utilisée pour manipuler toutes les **chaînes de caractères** nécessaires dans le jeu : titres de menu, messages, score à l'écran, etc.
- Elle offre des opérations simples et sûres sur les textes (concaténation, formatage, recherche), essentielles pour l'interface utilisateur.

4. Chrono

- La bibliothèque std::chrono est utilisée pour mesurer avec précision le temps écoulé pendant la partie.
- Elle permet de suivre exactement combien de temps le joueur est resté en jeu avant de perdre, ce qui est ensuite affiché à l'écran de fin de partie.

L'utilisation conjointe de **SFML** pour la gestion multimédia et de la **STL** pour les opérations de base a permis de construire une application à la fois **performante**, **organisée** et **facile à maintenir**. Ces bibliothèques offrent un socle solide pour un projet de jeu 2D, en simplifiant le développement tout en garantissant une excellente interactivité et une bonne extensibilité du code.

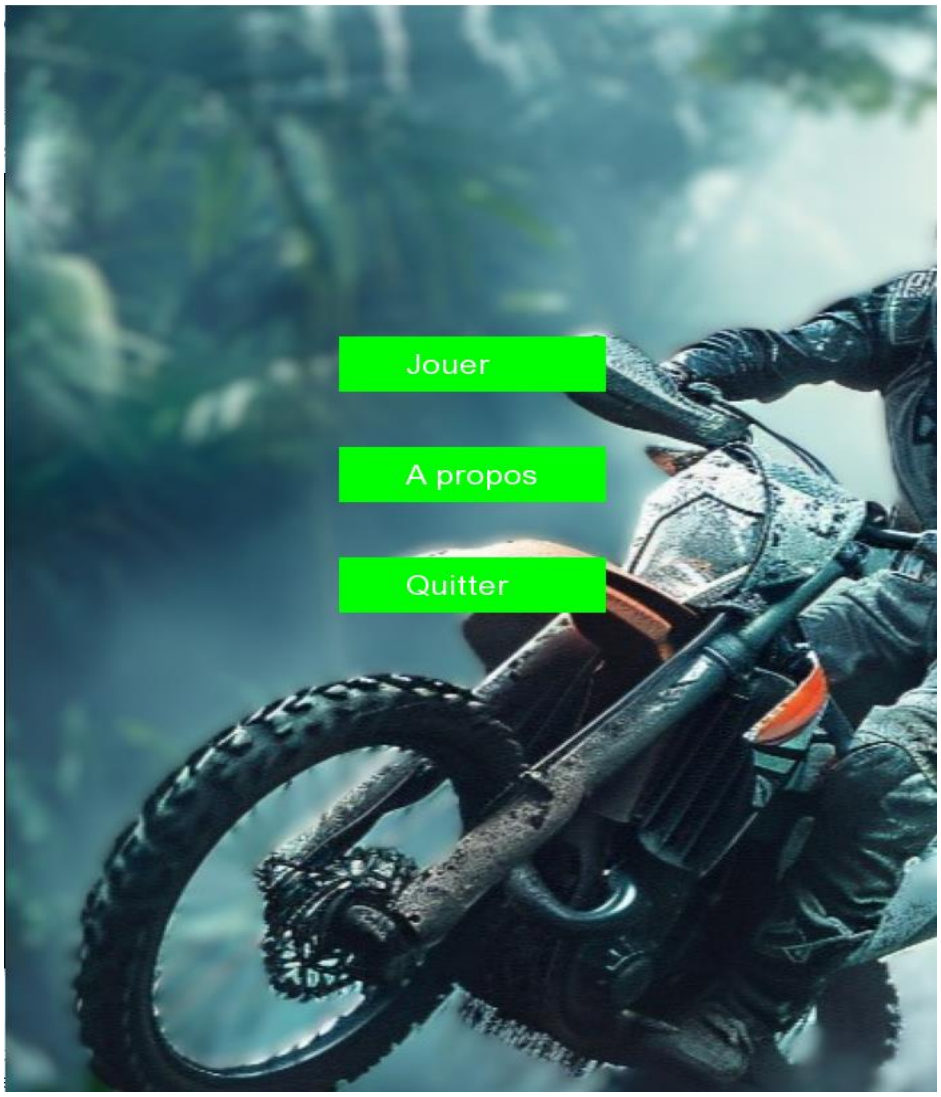
7. Interface graphique

L'interface graphique constitue un pilier fondamental du jeu de course à vélo, car elle représente l'environnement dans lequel l'utilisateur interagit avec l'application. Elle a été conçue de manière intuitive, épurée et attrayante pour offrir une expérience fluide et immersive. Cette interface est rendue à l'aide de la bibliothèque SFML, qui permet de dessiner des objets, d'afficher des textes et de gérer les entrées utilisateur en temps réel.

L'ensemble des éléments graphiques a été pensé pour maintenir une cohérence visuelle et une lisibilité optimale. La piste, les obstacles, le personnage principal (la moto), les éléments décoratifs comme les arbres ou les étoiles, ainsi que les menus, contribuent à l'esthétique générale du jeu.

Cette section présente les différentes vues de l’interface, accompagnées de descriptions détaillées de chaque élément graphique utilisé dans le jeu.

Figure 1 : Menu principal

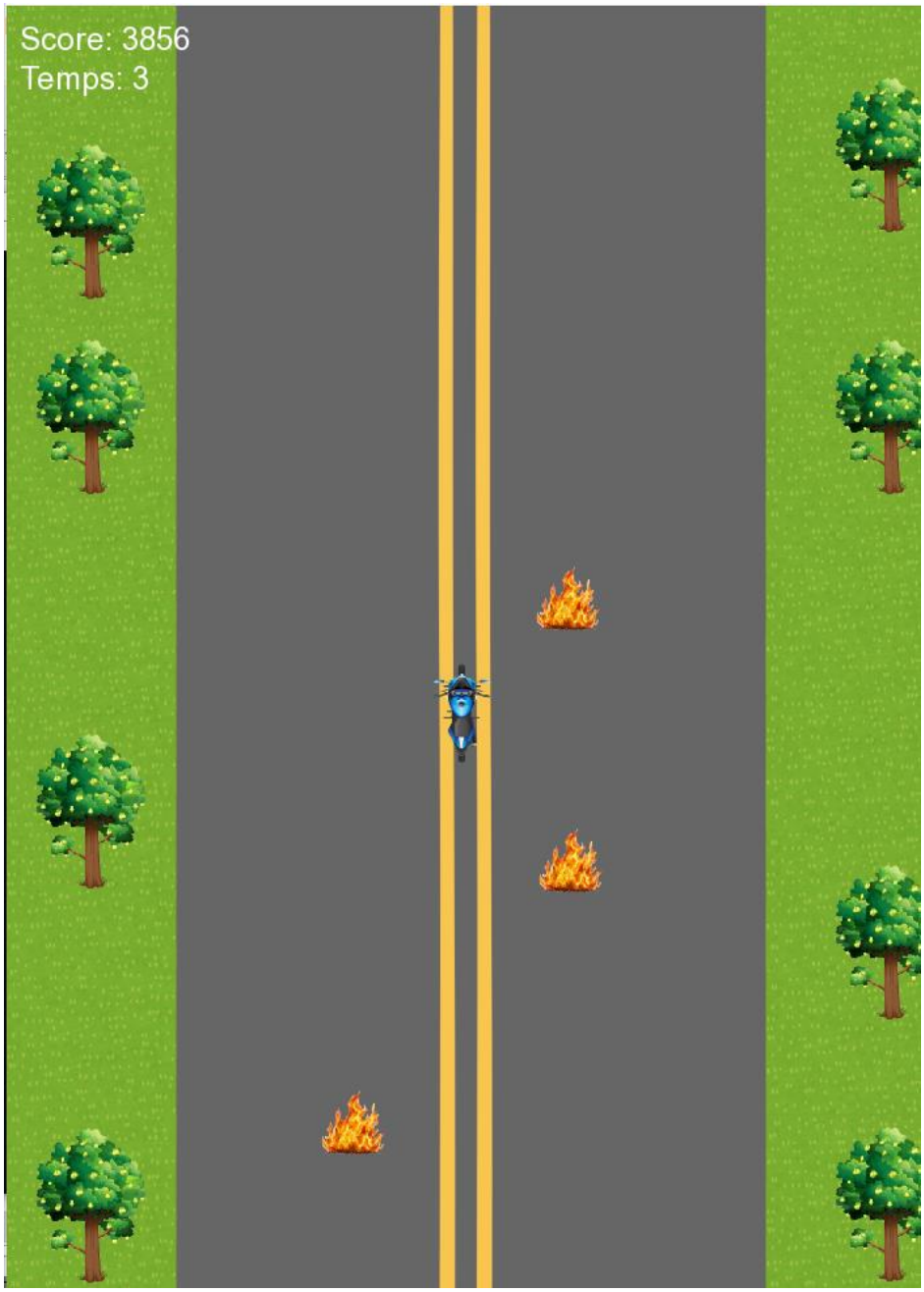


Légende : Interface de démarrage avec trois options cliquables : *Jouer*, *À propos* et *Quitter*.

Description :

Lorsque l'application est lancée, l'utilisateur est accueilli par un écran de menu simple et fonctionnel. Ce menu est interactif et entièrement contrôlable à l’aide de la souris. Il contient trois boutons bien distincts, permettant de naviguer rapidement vers les différentes fonctionnalités de l’application. En arrière-plan, une image d’une moto est affichée pour renforcer l’aspect visuel et donner un ton sportif et dynamique au jeu dès l’ouverture. Les boutons sont mis en évidence à l’aide de couleurs contrastées pour favoriser l’ergonomie. Le menu a été conçu pour être intuitif, même pour un joueur novice.

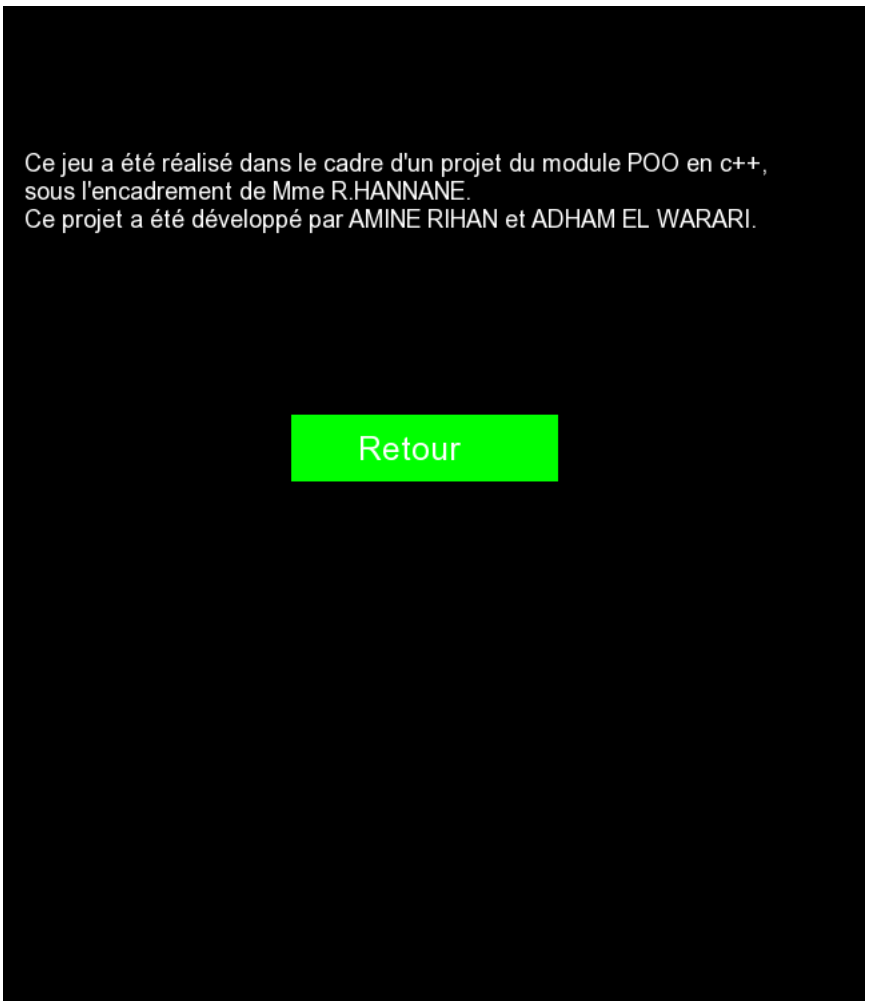
Figure 2 : Fenêtre de jeu



Légende : Le joueur contrôle une moto évoluant sur une piste jalonnée d'obstacles, avec un chronomètre affiché en haut à droite.

Description :
Une fois le bouton *Jouer* activé, le jeu se lance dans une nouvelle fenêtre. Le joueur incarne une moto représentée graphiquement par une forme SFML (souvent un rectangle stylisé), se déplaçant sur une piste droite. Des obstacles sous forme de feu sont répartis sur le trajet : le joueur doit les éviter pour ne pas perdre. Un arbre décoratif est placé sur le bord de la piste, ajoutant une touche réaliste et visuelle à l'environnement.
La direction de la moto est contrôlée par les touches directionnelles du clavier. L'ensemble de l'interface est fluide, fonctionnant à un taux élevé de rafraîchissement pour garantir une expérience agréable.

Figure 3 : Écran "À propos"

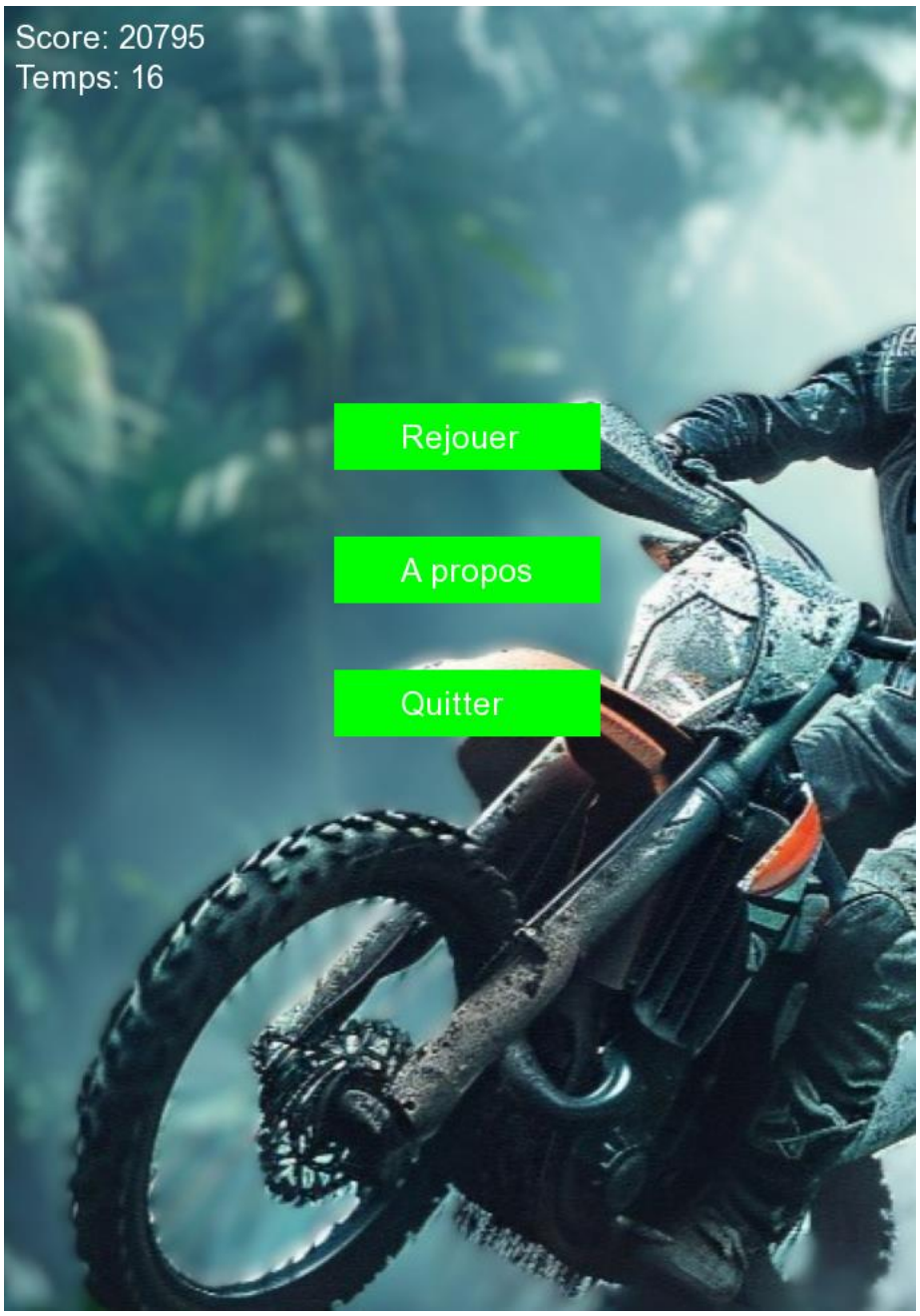


Légende : Présentation du projet et rappel des objectifs du jeu

Description :

Le bouton À propos donne accès à un écran d'information présentant brièvement le contexte du projet . Cette section affiche notamment que ce projet a été réalisé dans le cadre du module programmation orientée objet en c++ par AMINE RIHAN ET ADHAM EL WARARI , sous l'encadrement de Mme R.HANNANE.

Figure 4 : Fin de partie et menu après défaite



Légende : Affichage du score et du temps écoulé après une défaite par collision

Description :

Lorsque le joueur entre en collision avec un obstacle, la partie prend fin immédiatement. Un écran de fin de partie s'affiche alors, indiquant deux informations essentielles :

- Le score final du joueur, calculé en fonction des bonus collectés ou de la progression dans le jeu.
- Le temps total passé dans la partie, mesuré par un chronomètre actif depuis le début du jeu.

Ce temps n'impose pas de limite pendant la partie, il sert uniquement à informer le joueur de la durée de sa session une fois la partie terminée.

Après cet écran, le menu principal réapparaît, avec une adaptation mineure : le bouton "Jouer" est remplacé par "Rejouer", ce qui permet au joueur de relancer rapidement une nouvelle partie, sans repasser par toutes les étapes du menu initial.

Éléments graphiques additionnels

En plus des vues principales décrites ci-dessus, plusieurs objets décoratifs et interactifs ont été intégrés dans l'interface pour enrichir l'expérience visuelle et ludique :

- **Moto (joueur)** : Élément central du gameplay, la moto est contrôlée par le joueur et sert de repère principal à l'écran. Elle est dessinée avec des formes simples mais facilement identifiables.

- **Obstacles (Feu)** : Représentés par des rectangles rouges ou des sprites stylisés, ces obstacles sont positionnés de manière stratégique sur la piste. Leur rôle est d'augmenter la difficulté du jeu tout en stimulant la réactivité du joueur.

- **Piste** : La piste constitue l'aire de déplacement de la moto. Délimitée graphiquement, elle donne l'illusion d'un chemin rectiligne à parcourir, tout en imposant des limites de déplacement.

- **Arbre** : Placé sur les bords de la piste, l'arbre joue un rôle purement esthétique, contribuant à rendre le décor plus réaliste et moins monotone.

- **Étoile (Bonus)** : Ces objets peuvent être collectés volontairement par le joueur. Ils offrent des points supplémentaires et incitent à prendre des risques. Leur collecte est optionnelle, mais peut influencer fortement le score final.

8. Méthodologie de développement

Le développement de ce jeu de course à vélo a suivi une méthodologie structurée en plusieurs étapes progressives. Chaque phase a été planifiée afin de garantir un avancement fluide, cohérent, et conforme aux objectifs pédagogiques et techniques du projet. Le travail a été mené en binôme, en mettant en œuvre des pratiques de développement logiciel centrées sur la modularité, l'itérativité et la qualité du code.

8.1. Étapes suivies

1. **Définition des objectifs et contraintes**
   La première étape a consisté à clarifier les attentes du projet : créer un jeu simple mais fonctionnel intégrant des mécanismes de base (déplacement, collisions, score, interface graphique et sonore). Les contraintes de temps, d'outils (utilisation de SFML et du C++) ainsi que les capacités techniques de l'environnement de développement ont également été définies dès le début.

2. **Maquettage du menu et de la fenêtre de jeu**
   Avant d'écrire le code, une maquette simple a été conçue sur papier pour visualiser les interfaces du jeu : le menu principal, l'écran de jeu, l'écran À propos, et la fenêtre de fin de partie. Ce maquettage a facilité l'organisation des éléments graphiques à coder plus tard.

3. **Développement de la logique du vélo et des collisions**
   La logique de base du gameplay a été implémentée en priorité : déplacement fluide de la moto avec les touches directionnelles, détection des collisions avec les murs, et gestion de

la fin de partie en cas d'échec. Cette étape a représenté le cœur du développement fonctionnel du jeu.

4. **Intégration du menu interactif avec la souris**
   Une fois les fondations du jeu en place, l'interface utilisateur a été enrichie avec un menu principal interactif. Des boutons cliquables ont été ajoutés, permettant à l'utilisateur de lancer une partie, de consulter les règles ou de quitter le jeu.

5. **Intégration du son de fond avec SFML**
   L'immersion du joueur a été renforcée grâce à l'ajout d'une musique de fond, jouée en boucle via la classe sf::Music de SFML. Cela a nécessité l'intégration des fichiers .ogg dans le dossier du projet, ainsi que leur gestion dans le code.

6. **Ajout de la logique score/temps**
   Le jeu a été enrichi d'un système de score ainsi que d'un chronomètre affichant la durée de la partie. Le score augmente en fonction des bonus collectés pendant le jeu, ce qui incite le joueur à explorer et à interagir avec les éléments du décor. Le temps, mesuré à l'aide de la bibliothèque std::chrono, correspond à la durée totale que le joueur a passée en jeu avant la défaite. Ce chronomètre n'impose aucune limite, il sert uniquement à fournir un retour informatif sur la performance du joueur à la fin de chaque partie.

7. **Tests fonctionnels et itératifs**
   À chaque étape, des tests manuels ont été réalisés pour valider le bon fonctionnement des fonctionnalités ajoutées. Le développement s'est fait par cycles courts : ajout d'une fonctionnalité, test immédiat, correction des éventuels bogues, puis passage à la suivante.

8. **Nettoyage et modularisation du code**
   Une fois toutes les fonctionnalités stabilisées, le code a été réorganisé et commenté. Les fonctions ont été séparées selon leur rôle (affichage, gestion des événements, logique de jeu, etc.) afin d'améliorer la lisibilité et la maintenabilité du projet. Les classes ont été utilisées pour structurer les entités principales (jeu, joueur, obstacles…).

8.2. Stratégies de test

Pour assurer la stabilité du jeu et une expérience fluide, plusieurs tests ont été réalisés manuellement à différents niveaux :

- **Tests de déplacement du vélo**
  Vérification de la fluidité des mouvements avec les touches directionnelles. Des tests ont été faits pour s'assurer que le vélo reste dans les limites de la piste et qu'il répond correctement aux entrées utilisateur.

- **Tests de collision avec les murs**
  Scénarios variés de collision ont été simulés pour garantir que la partie se termine correctement en cas d'impact. Cela inclut des tests avec plusieurs murs, disposés de façon aléatoire ou rapprochée.

- **Test des clics dans le menu**
  Des tests ont été réalisés sur les différentes zones interactives du menu pour garantir que

chaque bouton fonctionne comme attendu : démarrer une partie, afficher les règles, quitter l'application.

- **Vérification du son et de la boucle musicale**
  Les sons ont été testés pour s'assurer qu'ils se jouent au bon moment, sans coupure ni décalage, et que la boucle musicale fonctionne correctement pendant toute la durée du jeu.

9. Limites actuelles

Malgré les fonctionnalités principales mises en œuvre avec succès dans ce projet de jeu de course à vélo, certaines limitations persistent, réduisant l'évolutivité et la richesse de l'expérience de jeu. Ces points faibles sont listés ci-dessous dans une optique d'amélioration future.

• Un seul niveau de jeu

Actuellement, le jeu ne propose qu'un seul et unique niveau. Le joueur rencontre donc toujours le même environnement, avec les mêmes mécaniques et obstacles à chaque partie. Cela limite fortement la rejouabilité et la diversité des défis proposés. Il serait pertinent d'intégrer des niveaux progressifs avec des difficultés croissantes, des obstacles variés, voire des changements de décors ou de conditions de jeu.

• Absence de système de sauvegarde du score

Le score obtenu à la fin d'une partie n'est pas enregistré d'une session à l'autre. Ainsi, il est impossible pour le joueur de suivre ses performances sur le long terme, ou de comparer ses résultats avec d'autres. Un système de sauvegarde locale ou même de classement en ligne pourrait apporter un aspect compétitif et incitatif.

• Graphismes minimalistes basés sur des formes géométriques

Les éléments visuels du jeu sont volontairement simples, reposant principalement sur l'utilisation de formes géométriques (rectangles, cercles) fournies par SFML. Bien que cette simplicité facilite le développement et la lisibilité, elle nuit quelque peu à l'immersion et à l'esthétique globale. L'ajout de sprites détaillés, d'animations, ou de textures enrichirait visuellement l'interface et rendrait le jeu plus attrayant.

• Sons basiques et non variés

L'ambiance sonore actuelle est assurée par une seule musique de fond, jouée en boucle. Aucun effet sonore supplémentaire (sons d'alerte, collisions, collecte de bonus, etc.) n'est encore intégré. Cela réduit la richesse sonore et peut rendre l'expérience monotone à la longue. L'introduction d'effets contextuels sonores et de musiques dynamiques améliorerait considérablement l'ambiance immersive.

• Absence de progression ou de niveaux multiples

Le jeu ne propose pas encore de système de progression d'un niveau à un autre. Il n'y a pas de transition ou de montée en difficulté après avoir terminé un niveau. Cela signifie que le joueur recommence toujours au même point, sans réelle évolution dans le gameplay. La mise en place d'un système de niveaux, de paliers ou de missions pourrait dynamiser le déroulement du jeu et maintenir l'engagement du joueur plus longtemps.

10. Améliorations possibles

Bien que le jeu fonctionne bien dans son état actuel, plusieurs pistes d'amélioration existent pour le rendre plus captivant, fluide et durable. Ces améliorations peuvent concerner aussi bien l'aspect technique que l'expérience utilisateur, en intégrant de nouvelles fonctionnalités et en affinant certains éléments du gameplay. Voici quelques améliorations qui pourraient enrichir l'expérience de jeu :

1. Ajout de niveaux supplémentaires

Actuellement, le jeu propose un seul niveau, ce qui peut rapidement réduire l'intérêt du joueur à long terme. Afin d'améliorer l'expérience et de renforcer la rejouabilité, il serait pertinent d'ajouter des **niveaux supplémentaires**. Chaque niveau pourrait introduire des défis uniques, offrant ainsi une expérience de jeu diversifiée.

Les niveaux pourraient progressivement augmenter en difficulté, avec des obstacles de plus en plus complexes et des parcours de plus en plus difficiles à naviguer. Par exemple, au fur et à mesure que le joueur progresse, il pourrait rencontrer des **obstacles mobiles**, des **zones de collision plus difficiles**, ou des **changements de perspective** tels que des virages serrés ou des pentes. De nouvelles mécaniques de jeu, comme des terrains glissants ou des ennemis à éviter, pourraient également être introduites pour enrichir l'expérience.

L'ajout de ces niveaux permettrait également de **sauvegarder la progression** du joueur entre les niveaux, garantissant ainsi une expérience continue. Cela offrirait aux joueurs un véritable challenge, tout en augmentant la durée de vie du jeu. Une telle fonctionnalité augmenterait considérablement la rejouabilité, en permettant au joueur de débloquer de nouveaux défis et de revenir jouer pour tester ses compétences sur des parcours de plus en plus complexes.

2. Système de sauvegarde du score

Le jeu affiche actuellement le score final à la fin de chaque partie, mais ce score est réinitialisé dès que le joueur relance une nouvelle session. Pour **rendre le jeu plus compétitif et motivant**, un **système de sauvegarde des meilleurs scores** pourrait être mis en place. Ce système permettrait de **conserver les meilleurs scores**, soit localement sur le dispositif du joueur, soit sur un serveur en ligne pour faciliter les classements mondiaux.

L'ajout d'un **tableau des scores** dans le menu principal permettrait aux joueurs de suivre leurs progrès au fil du temps et de comparer leurs performances avec celles des autres joueurs. Il pourrait également y avoir un mécanisme permettant de relancer une partie à un niveau de difficulté plus élevé, ce qui encouragerait les joueurs à se dépasser pour tenter de battre leurs précédents records.

Cette fonctionnalité serait particulièrement bénéfique pour les joueurs qui aiment rivaliser, améliorer leurs performances et atteindre des objectifs personnels. Elle apporterait également un aspect social au jeu, permettant de partager et comparer les résultats.

3. Ajout d'un tutoriel interactif

Un autre ajout majeur pour améliorer l'expérience de jeu serait l'intégration d'un **tutoriel interactif**. Ce tutoriel pourrait être visible lors du **premier lancement du jeu**, ou accessible à tout moment depuis le menu principal. Son objectif serait de **guider le joueur dans sa prise en main** en expliquant de manière claire et interactive les commandes, les objectifs et les mécaniques principales du jeu.

Le tutoriel pourrait se dérouler en plusieurs étapes, chacune abordant une fonctionnalité essentielle du jeu. Par exemple, après avoir expliqué les contrôles de la moto, le tutoriel pourrait inviter le joueur à naviguer sur la piste tout en évitant des obstacles (comme les feux) et en collectant des étoiles. Des conseils visuels, comme des flèches ou des zones surlignées, guideraient l'utilisateur à chaque étape.

L'interactivité serait un élément clé de ce tutoriel : au lieu de simplement fournir des instructions textuelles, le joueur serait invité à effectuer certaines actions pratiques pour mieux comprendre le gameplay. Cela permettrait non seulement de rendre l'apprentissage plus ludique et engageant, mais aussi d'assurer que le joueur assimile réellement les bases avant de se lancer dans le jeu complet.

11.Conclusion générale

Le projet de jeu de course à vélo développé avec SFML offre une expérience simple et engageante, tout en démontrant une bonne utilisation des bibliothèques graphiques et sonores. Depuis la conception de l'interface graphique jusqu'aux mécaniques de gameplay, chaque élément a été soigneusement pensé pour offrir une expérience fluide et immersive. Le jeu inclut un menu interactif, un système de chronométrage, et des obstacles à éviter, créant ainsi une dynamique de jeu excitante pour l'utilisateur.

Le développement a suivi une méthodologie structurée, depuis la définition des objectifs initiaux jusqu'aux tests fonctionnels. L'intégration des fonctionnalités s'est faite de manière itérative, permettant d'ajouter progressivement des éléments tout en garantissant leur bon fonctionnement. Les étapes de développement, comme l'ajout des événements interactifs et de la gestion du son de fond, ont permis de renforcer l'immersion dans le jeu.

Cependant, certaines limites actuelles subsistent, notamment l'absence de niveaux supplémentaires, de sauvegarde du score, et de graphismes plus détaillés. De plus, bien que le jeu soit fonctionnel, il reste encore des possibilités d'amélioration pour offrir une meilleure expérience à long terme. L'ajout de nouveaux niveaux et d'un système de sauvegarde des meilleurs scores permettrait de renforcer l'aspect compétitif et d'encourager la rejouabilité. De plus, un tutoriel interactif pour guider les nouveaux joueurs dans la prise en main du jeu serait un ajout précieux.

En conclusion, bien que ce projet soit un bon point de départ, de nombreuses améliorations peuvent être apportées pour enrichir l'expérience utilisateur et accroître la durée de vie du jeu. L'ajout de fonctionnalités telles que des niveaux variés, un tableau des scores, ainsi qu'un tutoriel interactif contribuerait à rendre le jeu plus captivant et accessible.