Université : Faculté des Sciences Semlalia Marrakech FSSM

Année Universitaire : 2024 – 2025

Mini-Projet : Application de jeu de course à vélo :

Langage utilisé : C++ (avec SFML)

Module : Programmation Orientée Objet – C++

Réalisé par :

Nom Prénom	Email Académique
Mahmoud Moukouch	m.moukouch2471@uca.ac.ma
Mohamed lakhdar	m.lakhdar4918@uca.ac.ma

Encadré par :

Prof. HANNANE Rachida

1.Introduction

La programmation orientée objet (POO) est une méthode clé pour structurer des applications informatiques de manière modulaire et évolutive. Ce projet a pour objectif de créer un jeu vidéo simple en C++ en utilisant la bibliothèque graphique SFML. Le jeu permettra aux utilisateurs de vivre une expérience interactive, où la logique de programmation et la gestion des éléments visuels se rencontrent.

L'idée principale est de concevoir un jeu dans lequel le joueur interagit avec un personnage ou un objet à l'écran, tout en respectant une série de contraintes et en relevant des défis. Cela permet non seulement d'apprendre à organiser un code de manière propre et réutilisable grâce à la POO, mais aussi de comprendre comment lier l’aspect visuel à la logique du jeu.

Ce projet est l'occasion de se familiariser avec la gestion des événements en temps réel, les manipulations graphiques, et la structuration du programme avec des concepts comme l'encapsulation, l’héritage, et la création de classes. À travers cette démarche, l'objectif est de mettre en pratique des compétences essentielles tout en créant une application interactive et fonctionnelle.

Ce document a pour but de détailler les étapes de conception et de développement du jeu, en expliquant les choix techniques et les outils utilisés.

2. Spécification des besoins

Ce chapitre a pour objectif de définir les principales fonctionnalités attendues pour le jeu de course à vélo, tout en détaillant les critères techniques et fonctionnels nécessaires à sa réalisation. Il établit les bases sur lesquelles reposera la conception du jeu, en clarifiant les objectifs à atteindre avant d’entamer le développement.

a. Fonctionnalités principales

L'application devra inclure les fonctionnalités suivantes :

- Un menu principal visible dès le démarrage, offrant trois options :
    - **Jouer** : Permet de commencer une nouvelle partie.
    - **À propos** : Affiche des informations sur le jeu et son processus de développement.
    - **Quitter** : Ferme l'application.
- Le jeu solo, où un seul joueur contrôle le vélo à l'aide des touches directionnelles du clavier (haut, bas, gauche, droite | W, S, A, D).
- Un **chronomètre** avec une limite de temps, qui décompte automatiquement. Le but est d'atteindre l'arrivée avant que le temps ne soit écoulé.
- **Gestion des collisions** :
    - Si le vélo heurte un obstacle, la partie est immédiatement perdue.
    - Le vélo doit toujours rester à l'intérieur des limites de la piste et ne pas sortir de son tracé.
- Un **affichage dynamique**, où le vélo, les obstacles, ainsi que le chronomètre se mettent à jour en temps réel pendant la partie.

Here's a rephrased version of the section with the technical constraints and user interaction:

b. Contraintes techniques

- **SFML** sera utilisé pour gérer l'affichage en 2D, les événements du clavier, le chronomètre et la détection des collisions.
- L'application sera conçue en suivant les principes de la **programmation orientée objet (POO)**, en structurant le code autour de classes pour assurer la modularité et l'encapsulation.
- **Performance** : L'application doit être fluide, sans latence, avec une réactivité optimale.
- **Design personnalisé** : Chaque équipe devra proposer un design graphique unique et original pour l'interface du jeu.

c. Interaction utilisateur

- Le menu doit offrir une **navigation intuitive**, permettant à l'utilisateur de sélectionner une option à l'aide des touches directionnelles du clavier.
- Un retour visuel immédiat sera affiché lors des déplacements du vélo, afin d'assurer une bonne réactivité.

- Des **messages clairs** seront affichés pour indiquer les résultats de la partie, que ce soit en cas de victoire ou de défaite.
- Le joueur pourra **relancer une partie** sans avoir à redémarrer l'application, facilitant ainsi les tests et la rejouabilité.

3. Conception et Modélisation (Structure des Classes)

Le jeu est structuré selon les principes de la programmation orientée objet, ce qui permet d'organiser le code de manière modulaire et de séparer clairement les différentes responsabilités. Chaque composant du jeu est encapsulé dans une classe spécifique, ce qui simplifie les modifications futures et assure une maintenance plus efficace. Cette approche permet également une meilleure évolutivité du projet. Ci-dessous, nous présentons un aperçu des classes principales, de leurs fonctions clés et des méthodes associées.

a. Classe "Player"

Rôle (Joueur) :

La classe Player gère toutes les interactions du joueur avec le jeu, y compris le mouvement du vélo, la gestion de la vitesse, et les interactions avec les obstacles. Elle permet au joueur de se déplacer à gauche et à droite sur la piste, d'accélérer avec un système d'endurance et de gérer des événements tels que les collisions et les sauts. Elle met également à jour la position du vélo et de son ombre, en tenant compte des entrées clavier.

Attributs principaux :

- sf::Sprite sprite : Représentation visuelle du vélo du joueur.
- sf::Sprite shadow : Ombre du vélo pour un effet visuel amélioré.
- int playerLane : Voie actuelle du joueur (de 0 à 3).
- float stamina : Endurance du joueur, utilisée pour accélérer.
- float playerWorldSpeed : Vitesse actuelle du vélo, affectée par les entrées du joueur.
- sf::Clock invincibilityClock : Chronomètre pour gérer la durée de l'invincibilité.
- bool invincible : Indicateur d'invincibilité du joueur.
- bool triedWhileExhausted : Indicateur si le joueur a essayé d'accélérer sans suffisamment d'endurance.

Méthodes principales :

- `void Player::resetPosition` : Réinitialise la position du joueur sur la piste en fonction des paramètres de la route et de la fenêtre de jeu.
- `void Player::update(float dt, sf::Sound& tiredSound)` : Met à jour l'état du joueur, gérant l'accélération, le freinage, la régénération d'endurance et les effets sonores si l'endurance est insuffisante.
- `bool Player::isInvincible()` : Retourne vrai si le joueur est invincible.
- `void Player::startInvincibility()` : Active l'invincibilité du joueur.
- `void Player::draw(sf::RenderWindow& window)` : Affiche le vélo et son ombre, avec un effet de clignotement si le joueur est invincible.
- `moveLeft()` et `moveRight(int LANES)`: Déplace le joueur à gauche ou à droite sur la piste, en respectant les limites de la voie.

Rôle (Vélo) :

La classe Player représente le vélo du joueur et permet la gestion de ses déplacements sur la route, les actions comme l'accélération et le freinage, ainsi que la détection des collisions avec les obstacles. Elle gère également l'affichage visuel du vélo et la mise à jour en fonction des entrées du joueur.

Interaction avec le jeu :

- Le joueur peut se déplacer entre les différentes voies de la route.
- Les touches W, S et Space contrôlent respectivement l'accélération, le freinage et le saut du vélo.
- Les collisions avec les obstacles, comme les rochers, sont détectées, et le jeu prend en compte les vies du joueur.

b. Classe "Eplayers"

Rôle :

La classe Obstacle représente un obstacle dynamique sur la route, tel qu'un rocher ou tout autre objet que le joueur doit éviter. Ces obstacles apparaissent aléatoirement sur la route et se déplacent vers le bas, représentant un défi que le joueur doit surmonter. Lorsqu'une collision avec le vélo du joueur est détectée, l'obstacle entraîne une perte de vie.

Attributs principaux :

- std::vector<sf::Sprite> obstacles → Liste des obstacles générés dans le jeu.
- std::vector<sf::Texture> textures → Contient les textures des différents types d'obstacles disponibles.

Méthodes principales :

- void spawn(float roadTextureWidth, float padLeft, float padRight, int LANES, sf::RenderWindow& window)
  Cette méthode génère de nouveaux obstacles de manière aléatoire sur la route. Un obstacle est placé dans une voie choisie aléatoirement et à une position verticale juste au-dessus de l'écran.
- bool update(float obstacleSpeed, bool braking, sf::RenderWindow& window, sf::Sprite& player, sf::Sound& crashSound, int& lives, int& score)
  Cette méthode met à jour la position de chaque obstacle, déplaçant les obstacles vers le bas de l'écran. Elle détecte les collisions avec le joueur, joue un son de collision, et diminue les vies du joueur. Elle gère également l'élimination des obstacles lorsque ceux-ci passent sous l'écran ou lorsque des collisions sont détectées.
- void afficher(sf::RenderWindow& window)
  Affiche l'obstacle à l'écran, incluant un effet d'ombre pour chaque obstacle, offrant une meilleure visibilité pendant le jeu.

Interaction avec le jeu :

- Les obstacles apparaissent aléatoirement dans différentes voies de la route.
- Ils se déplacent vers le bas de l'écran, représentant un danger à éviter.
- En cas de collision entre un obstacle et le vélo du joueur, une perte de vie est enregistrée et l'obstacle est supprimé de la scène.

c. Classe “Crono”

Rôle :

La classe Crono gère le chronomètre utilisé pendant la partie, affichant le temps restant à l'écran avec une précision au centième de seconde. Elle est utilisée pour suivre la durée du jeu ou d’un niveau, et met à jour l’affichage du temps restant à chaque frame.

Attributs principaux :

- sf::Clock clock_ → Horloge interne utilisée pour mesurer le temps écoulé depuis le début du chronomètre.
- sf::Text text_ → Texte SFML affichant le temps restant à l'écran.
- sf::Font font_ → Police utilisée pour le texte du chronomètre.
- const float DURATION → Durée totale du chronomètre en secondes (fixée au départ, comme constante).

Méthodes principales :

- Crono(const sf::Font& font, unsigned characterSize)
  Constructeur qui initialise la police et les caractéristiques du texte (taille, couleur, etc.), et démarre le chronomètre.
- void restart()
  Redémarre le chronomètre, réinitialisant l'horloge interne.
- void update()
  Mise à jour du chronomètre (bien que cette méthode soit actuellement vide, elle peut être utilisée pour des ajustements comme le formatage ou des limites).
- bool isFinished() const
  Vérifie si la durée du chronomètre a été atteinte (temps écoulé supérieur ou égal à la durée définie).
- float getRemaining() const
  Renvoie le temps restant en secondes. Si le temps est écoulé, retourne 0.
- void draw(sf::RenderTarget& target)
  Affiche le chronomètre sur l'écran. Le temps restant est formaté en secondes entières et positionné dans le coin supérieur droit de la fenêtre de jeu.

Interaction avec le jeu :

- Le chronomètre commence dès le démarrage du jeu ou du niveau.
- Le temps restant est mis à jour et affiché sur l'écran à chaque frame.
- Lorsque le chronomètre atteint zéro, il indique que la durée est expirée (ce qui peut entraîner la fin du jeu ou un niveau).

d. Classe “Collectables”

Rôle :

La classe Collectables gère la génération, mise à jour et affichage des objets collectables (bouteilles et pièces) dans le jeu. Ces objets peuvent être ramassés par le joueur pour augmenter son score ou sa barre d'endurance (stamina).

Attributs principaux :

- sf::Texture bottleTexture → Texture utilisée pour afficher les bouteilles.
- sf::Texture coinTexture → Texture utilisée pour afficher les pièces.
- std::vector<sf::Sprite> bottles → Liste des bouteilles présentes dans la scène.
- std::vector<sf::Sprite> coins → Liste des pièces présentes dans la scène.

Méthodes principales :

- Collectables(const sf::Texture& bottleTexture, const sf::Texture& coinTexture)
  Constructeur qui initialise les textures des objets collectables (bouteilles et pièces).
- void spawnBottle(float roadTextureWidth, float padLeft, float padRight, int LANES, sf::RenderWindow& window, const std::vector<sf::Sprite>& obstacles, const std::vector<sf::Sprite>& coins)
  Gère la génération aléatoire des bouteilles sur la route. Vérifie qu'il n'y a pas de chevauchement avec les autres objets collectables ou les obstacles avant de les ajouter à la scène.
- void spawnScoreCoin(float roadTextureWidth, float padLeft, float padRight, int LANES, sf::RenderWindow& window, const std::vector<sf::Sprite>& obstacles, const std::vector<sf::Sprite>& bottles)
  Gère la génération aléatoire des pièces de score sur la route. Vérifie qu'il n'y a pas de chevauchement avec les autres objets collectables ou les obstacles avant de les ajouter à la scène.
- void updateBottles(float worldSpeed, sf::RenderWindow& window, sf::Sprite& player, float& stamina, const float MAX_STAMINA, const float BOTTLE_STAMINA, sf::Sound& drinkSound)
  Met à jour la position des bouteilles et gère la logique lorsque le joueur ramasse une bouteille. Lorsqu'une bouteille est ramassée, l'endurance du joueur augmente et la bouteille est supprimée de la scène.

- void updateCoins(float worldSpeed, sf::RenderWindow& window, sf::Sprite& player, int& score, sf::Sound& coinSound)
  Met à jour la position des pièces et gère la logique lorsque le joueur ramasse une pièce. Lorsqu'une pièce est ramassée, le score du joueur augmente et la pièce est supprimée de la scène.

Interaction avec le jeu :

- Les objets collectables (bouteilles et pièces) apparaissent aléatoirement sur la route, à des intervalles définis par la probabilité de génération.
- Lorsqu'une bouteille ou une pièce entre en collision avec le joueur, l'endurance ou le score du joueur est mis à jour respectivement, et l'objet collecté est supprimé de la scène.
- Les objets collectables qui sortent de la fenêtre de jeu sont également supprimés.

4. Environnement de développement

Ce chapitre détaille les outils, les bibliothèques et les environnements utilisés pour créer, développer et tester le jeu de course à vélo.

a. Langage de programmation

Le jeu a été entièrement programmé en C++, un langage robuste et polyvalent qui est particulièrement adapté à la programmation orientée objet (POO). Parmi ses principaux atouts, on retrouve :

- Une gestion de la mémoire très fine, permettant d'optimiser les performances du jeu.
- Une architecture modulaire, facilitant l'organisation du code grâce à l'utilisation de classes et de structures claires.

b. Environnement de développement (IDE)

Le développement s'est fait dans **Visual Studio Code**, un éditeur de code léger et flexible, enrichi de plugins spécifiques pour C++, ce qui permet de bien gérer les projets comportant plusieurs fichiers.

Autres IDEs recommandés pour le C++ :

- **CodeBlocks** avec le compilateur g++ de GCC, un choix classique pour le développement C++.
- **CLion**, offrant des outils avancés et une gestion intégrée du projet.
- **Dev-C++**, adapté pour les projets à vocation pédagogique.

c. Bibliothèque graphique : SFML

Le jeu s'appuie sur **SFML** (Simple and Fast Multimedia Library), une bibliothèque C++ moderne pensée pour simplifier la création de jeux 2D et d'applications multimédia.

Fonctions principales de SFML dans le projet :

- Création et gestion de la fenêtre du jeu.
- Affichage des images et animations (comme le vélo, les obstacles, et le décor).
- Gestion des interactions via le clavier pour déplacer le vélo.
- Suivi du temps de jeu et affichage du chronomètre.
- Détection des collisions à l'aide de boîtes de détection rectangulaires.
- Rendu des textes à l'écran, tels que les messages et le temps écoulé, grâce à la gestion des polices de caractères.

Points forts de SFML :

- Une interface orientée objet simple à utiliser.
- Une documentation complète et bien structurée, facilitant son apprentissage.
- Compatible avec les principaux systèmes d'exploitation : Windows, Linux et macOS.
- Une parfaite intégration avec C++, permettant de tirer pleinement parti des capacités du langage pour les jeux 2D.

c. Bibliothèque graphique : SFML

Le projet utilise **SFML** (Simple and Fast Multimedia Library), une bibliothèque moderne en C++ qui facilite le développement d'applications multimédia, notamment les jeux 2D.

Principales fonctions de SFML utilisées dans ce projet :

- Création et gestion de la fenêtre du jeu.
- Affichage des sprites (vélo, obstacles, éléments du décor, etc.).
- Gestion des entrées clavier pour contrôler les déplacements du vélo.

- Mesure et affichage du temps écoulé via un chronomètre.
- Détection des collisions à l'aide de coordonnées rectangulaires.
- Lecture de fichiers de polices pour afficher des textes à l'écran (temps, messages divers, etc.).

Avantages de SFML :

- Une interface orientée objet simple à prendre en main.
- Une documentation complète et bien structurée.
- Compatible avec les principaux systèmes d'exploitation : Windows, Linux, et macOS.
- Intégration parfaite avec le langage C++, permettant de maximiser ses capacités dans le développement de jeux 2D.

d. Structure des fichiers

Pour garantir une organisation claire et une bonne séparation des responsabilités, le projet suit la structure suivante :

Fichiers sources et d'en-tête :

- **main.cpp** → Point d'entrée du programme, gestion de l'interface utilisateur et de la navigation.
- **startgame.cpp / startgame.h** → Logique du jeu, contrôle du vélo, définitions globales (dimensions, vitesses, limites, etc.).
- **rock.cpp / rock.h** → Définition des obstacles dans le jeu.
- **chronometer.cpp / chronometer.h** → Gestion du chronomètre pour suivre le temps de jeu.

Dossier "ressources/" :
Ce dossier contient les éléments nécessaires au jeu, tels que les sprites, les icônes et les fichiers audio.

e. Gestion du projet

Les outils et méthodes suivants ont été utilisés pour la gestion du projet :

- **Compilation** : Utilisation de **Makefiles** ou des options de compilation offertes par l'IDE pour automatiser le processus.
- **Débogage** : Analyse des erreurs via des messages de log et des tests réalisés dans la console pour identifier et résoudre les problèmes.
- **Tests manuels** : Vérification des fonctionnalités principales du jeu, comme le bon fonctionnement des menus, les déplacements du vélo, la gestion des collisions et le respect des limites de temps.

5. L'implémentation (Interfaces)

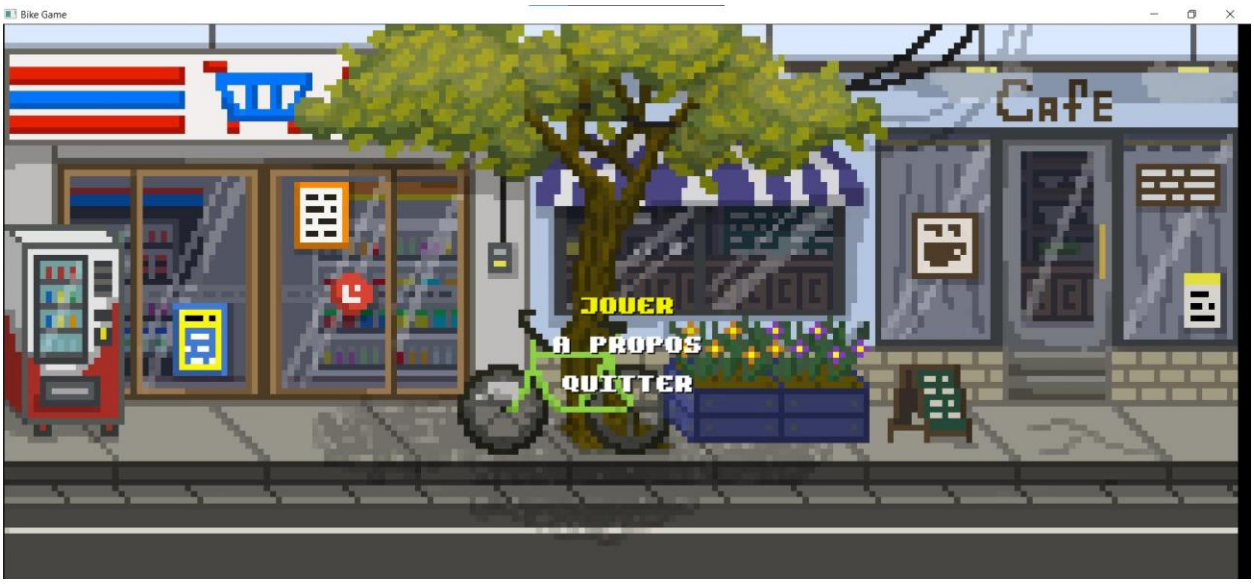
L’interface utilisateur du jeu de course à vélo a été pensée pour être à la fois simple, intuitive et immersive, tout en respectant les principes de conception ergonomique typiques des jeux 2D. Cette section présente les différentes interfaces utilisées tout au long du jeu.

a. Menu principal

Description :
Lors du lancement du jeu, l'utilisateur arrive sur un menu principal où trois options sont proposées :

- **Jouer** → Démarre une nouvelle partie.
- **À propos** → Affiche des informations sur le jeu et ses règles.
- **Quitter** → Ferme l’application.

Navigation :
Le joueur navigue entre ces options en utilisant les touches directionnelles. L'option sélectionnée est visuellement mise en évidence pour indiquer clairement la sélection en cours.

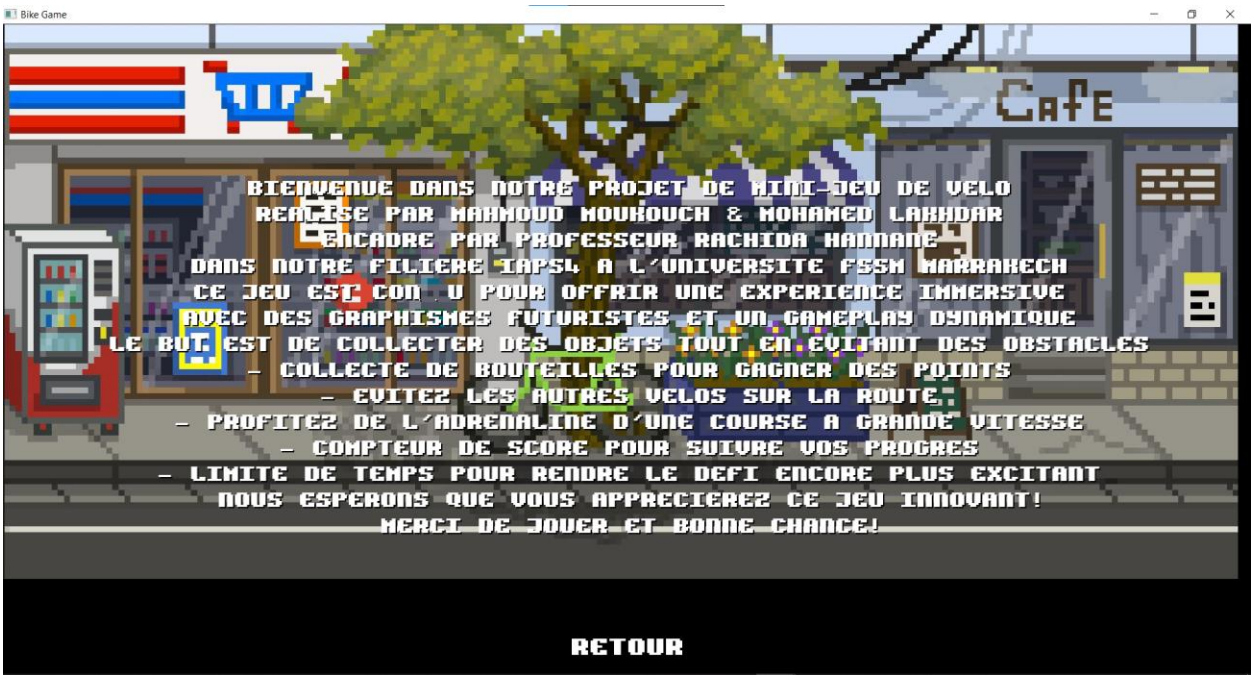


b. Écran "À propos"

Description :
Cet écran présente une explication rapide des règles du jeu, ses objectifs ainsi que les contrôles. Le texte défile de bas en haut, créant ainsi une animation fluide pour attirer l'attention du joueur.

Contenu typique :

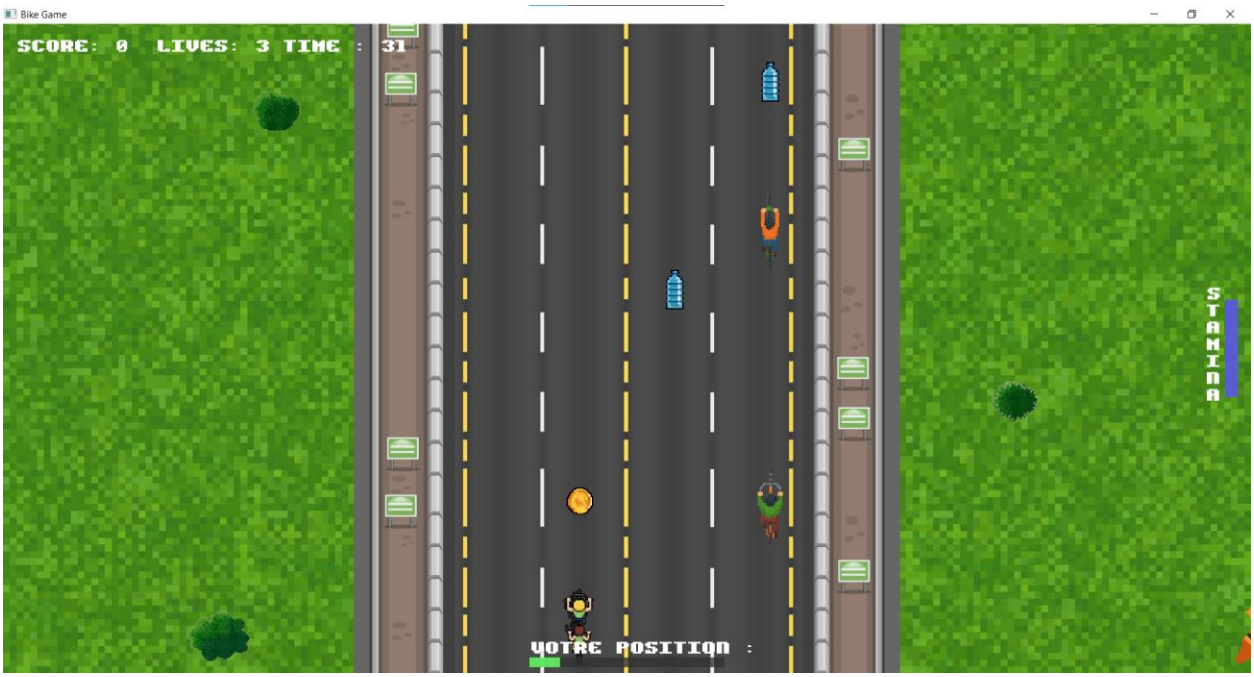
- But du jeu.
- Instructions pour les contrôles (touches directionnelles).
- Conditions de victoire et de défaite.
- Une button retour pour retourner au menu.



c. Fenêtre de jeu

Description :
Une fois la partie lancée, le joueur interagit avec l'interface principale du jeu, qui comprend :

- Un vélo contrôlé à l'aide du clavier (touches directionnelles).
- Des adversaires sous forme d'autres cyclistes, servant d'obstacles dynamiques à éviter.
- Un chronomètre visible en haut de l'écran pour suivre le temps de jeu.
- Un arrière-plan stylisé qui renforce l'immersion visuelle.

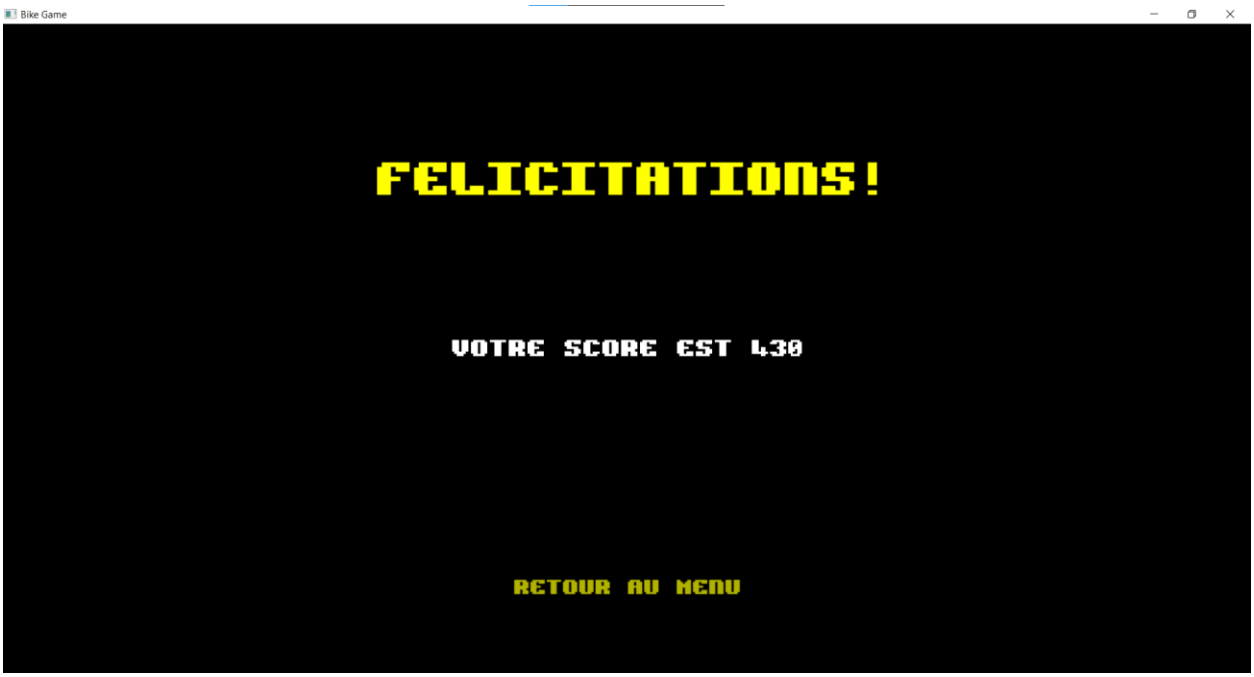


d. Écran de fin de partie

Description :
À la fin d'une partie, un message de **victoire** s'affiche à l'écran si le joueur atteint l'arrivée dans le temps imparti.

Le joueur peut alors choisir entre deux options :

- **Retourner au menu principal** → Revenir à l'écran d'accueil.



6. Conclusion

Ce mini-projet de création d'un jeu de course à vélo en C++ nous a offert une excellente opportunité de mettre en œuvre les principes de la programmation orientée objet et de découvrir la bibliothèque SFML, une solution efficace pour le développement de jeux 2D.

Tout au long du développement, nous avons appliqué plusieurs notions clés :

- Une conception modulaire avec une organisation structurée des fichiers et des classes.
- La gestion des événements et des interactions utilisateur, notamment à travers les commandes clavier.
- L'utilisation graphique de SFML pour l'intégration de sprites, textures et animations.
- Une structuration de code soignée, conforme aux bonnes pratiques d'encapsulation et de clarté.
- La mise en place d'une interface intuitive, alliant ergonomie et esthétique.

Ce projet nous a aussi permis de mieux appréhender les contraintes du développement de jeux, telles que la gestion du temps, des collisions, ou encore l'optimisation des performances.

Nous avons particulièrement apprécié la liberté de création offerte, qui nous a permis d'imaginer une interface originale et un gameplay en accord avec les objectifs fixés. Malgré

un temps de développement limité, nous avons adopté une approche souple, ajustant régulièrement notre code selon les tests et observations.

Références:

road : https://opengameart.org/content/2d-top-down-highway-background

player : https://chatgpt.com

eplayers :https://chatgpt.com

Finish Line: https://www.shutterstock.com/search/finish-line-pixel-art

font : https://www.dafont.com/pixelite.font

grass : https://www.craiyon.com/search/2d-pixel-art-grass

coins : https://www.dreamstime.com/gold-coin-bit-pixel-graphics-icon-pixel-art-style-game-assets-bit-sprite-isolated-vector-illustration-eps-gold-coin-bit-pixel-image231952453

bottle : https://www.shutterstock.com/search/plastic-bottle-pixel?image_type=illustration