MINI-PROJET EN C++

APPLICATION DE JEU DE COURSE À VÉLO

Réalisé par :

Noura Ait El Hadj
Moncef Aaouine

Encadré par :

Prof. R. HANNANE

Module : Programmation Orientée Objet (POO-C++

Année Universitaire 2024-2025

Date de remise : 10 mai 2025

PLAN :

I. Introduction :

Ce projet de développement d'un jeu de course à vélo en 2D, réalisé en C++ avec la bibliothèque SFML, s'inscrit dans le cadre de notre formation en Programmation Orientée Objet. Il a pour but de concilier apprentissage théorique et mise en pratique concrète en proposant une expérience ludique où le joueur contrôle un cycliste devant éviter des obstacles tout en gérant un chronomètre et un système de score. Ce projet présente un triple intérêt : pédagogique, technique et moderne.

Sur le plan **pédagogique**, il nous a permis d'appliquer les concepts fondamentaux de la POO - encapsulation, héritage et polymorphisme - dans un contexte réel de développement. La création de classes pour les différents éléments du jeu (vélo, obstacles, décor) et l'utilisation de mécanismes avancés comme le défilement parallaxe (créant l'illusion de profondeur) ont renforcé notre maîtrise du C++ moderne.

D'un point de vue **technique**, le projet représente une réelle avancée par rapport aux exercices initiaux. La gestion précise des collisions, l'implémentation d'une physique réaliste pour les sauts, et l'optimisation des performances graphiques ont constitué des défis stimulants. Le système de difficulté progressive, où la vitesse augmente avec le temps, ajoute une dimension stratégique au gameplay.

Le choix de **SFML** comme bibliothèque graphique s'est révélé judicieux pour ce projet. Son API moderne et performante, couplée à sa simplicité d'utilisation, nous a permis de nous concentrer sur la logique métier tout en bénéficiant de fonctionnalités avancées comme le double buffering ou la gestion des entrées. Notre environnement de développement, basé sur Visual Studio Code, CMake et Git, a favorisé une collaboration efficace et un workflow professionnel.

Les objectifs spécifiques du projet incluaient :

- ✓ Le développement d'une interface complète avec menu interactif
- ✓ L'implémentation de mécaniques de jeu robustes (mouvement, collisions, timer)
- ✓ Le respect strict des bonnes pratiques de POO
- ✓ L'exploitation optimale des capacités de SFML

Ce projet représente ainsi une synthèse réussie entre apprentissage académique et développement concret, tout en démontrant notre capacité à maîtriser des concepts complexes dans un cadre professionnel.

II. Conception du jeu :

2.1. Architecture générale :

Notre application suit une architecture modulaire bien organisée, séparant clairement les différents composants :

```
Projet/
├── 🗀 assets/          # Ressources du jeu
│   ├── 🗀 fonts/       # Polices d'écriture
│   ├── 🗀 images/      # Images utilisées
│   └── 🗀 textures/    # Textures graphiques
```

```
│
├── 🗁 build/           # Fichiers de compilation
│   ├── 🗁 CMakeFiles/
│   ├── cmake_install.cmake
│   ├── CMakeCache.txt
│   ├── Makefile
│   └── NewGame.exe      # Exécutable final
│
├── 🗁 include/         # Fichiers d'en-tête (.hpp)
│   ├── About.hpp
│   ├── Bike.hpp
│   ├── Constants.hpp
│   ├── Game.hpp
│   ├── Menu.hpp
│   ├── Obstacle.hpp
│   ├── Scores.hpp
│   ├── ScrollingBackground.hpp
│   └── Timer.hpp
│
└── 🗁 src/             # Code source (.cpp)
    ├── About.cpp
    ├── Bike.cpp
    ├── Game.cpp        # Boucle principale du jeu
    ├── main.cpp        # Point d'entrée
    ├── Menu.cpp
    ├── Obstacle.cpp    # Gestion des obstacles
    ├── Scores.cpp
    ├── ScrollingBackground.cpp
    └── Timer.cpp       # Gestion du chronomètre
```

2.2. Description des classes principales et relations :

✓ Description :

a. **Classe Game**

Rôle central :

Cette classe agit comme le cerveau de l'application, coordonnant tous les éléments du jeu. Elle

initialise la fenêtre graphique, gère les différents états (menu, jeu, scores) et supervise la boucle principale qui actualise le jeu 60 fois par seconde.

Fonctionnalités remarquables :

Contrôle la progression du jeu selon trois phases claires : affichage du menu, déroulement de la partie, affichage des résultats

Centralise la gestion des ressources graphiques et sonores

Assure la synchronisation entre les différents composants (vélo, obstacles, décor)

Gère les transitions fluides entre les différents écrans

b. **Classe Bike**

Représentation du joueur :
Le vélo constitue l'élément central que l'utilisateur contrôle. Sa modélisation intègre une physique réaliste pour des mouvements crédibles.

Mécaniques implémentées :

Déplacement latéral avec accélération progressive

Système de saut à double impulsion (simple et double saut)

Détection précise des collisions avec l'environnement

Animation fluide grâce à l'interpolation des positions

c. **Classe Obstacle**

Éléments de challenge :
Les obstacles représentent les principaux défis que le joueur doit surmonter, avec une génération aléatoire pour renouveler l'intérêt.

Caractéristiques techniques :

Apparition progressive avec augmentation de la fréquence

Déplacement synchronisé avec le décor pour une cohérence visuelle

Variété de formes et tailles pour diversifier le gameplay

Optimisation des performances via un pool d'objets réutilisables

d. **Classe Timer**

Gestion du temps :
Ce composant ajoute une dimension stratégique en imposant une limite temporelle.

Fonctionnalités innovantes :

Compte à rebours visible en permanence à l'écran

Adaptation dynamique de la difficulté selon le temps restant

Effets visuels pour marquer les derniers instants

Sauvegarde des meilleurs temps entre les parties

e. Classe ScrollingBackground

Immersion visuelle :

Le décor défilant crée une illusion de mouvement et de profondeur.

Techniques employées :

Utilisation du parallaxe pour plusieurs plans de profondeur

Bouclage parfait sans saccades visibles

Adaptation automatique à différentes résolutions

Chargement optimisé des textures

f. Classe Menu

Interface utilisateur :

Point d'entrée vers les différentes fonctionnalités du jeu.

Éléments clés :

Navigation intuitive au clavier ou à la souris

Effets visuels lors de la sélection des options

Accès rapide aux différents modes de jeu

Intégration harmonieuse avec le style graphique

g. Classe About (Informations)

Contenu pédagogique :
Gère l'ensemble des informations accessibles depuis le menu principal.

Éléments clés :

Affichage structuré des crédits et contributeurs

Pages d'instructions détaillées avec captures visuelles

Schéma interactif des commandes de jeu

Section dédiée aux mentions légales et licences

h. Classe Scores (Classement)

Gestion des performances :
Système complet d'archivage et de visualisation des résultats.

Fonctionnalités principales :

Sauvegarde sécurisée des scores avec horodatage

Classement intelligent par difficulté et mode de jeu

Affichage des records sous forme de tableau et graphiques

Système de récompenses visuelles pour les hauts scores

i. Classe Constants (Configuration)

Paramétrage centralisé :
Nœud de configuration de l'ensemble du projet.

Données managées :

Réglages graphiques et sonores prédéfinis

Constantes physiques du gameplay

Chemins d'accès aux ressources assets

Variables d'équilibrage du niveau de difficulté

✓ Les relations :

⇨ **Principes POO appliqués :**

Encapsulation : Protection des données via des membres privés (ex: *mVelocityY* dans Bike) avec accès contrôlé par getters/setters.

Polymorphisme : Redéfinition de méthodes comme *draw()* pour adapter le comportement selon les classes.

⇨ **Relations entre les calsses :**

Notre architecture utilise trois types fondamentaux de relations entre classes, chacune choisie pour répondre à des besoins spécifiques de conception.

L'héritage a été employé avec parcimonie, principalement pour intégrer les fonctionnalités de base de SFML. La classe ScrollingBackground hérite ainsi de *sf::Drawable*, ce qui lui permet de redéfinir la méthode de rendu graphique tout en respectant le contrat imposé par la bibliothèque. Ce choix respecte le principe de substitution de Liskov et maintient une cohérence avec l'écosystème SFML.

La composition représente la relation privilégiée pour les éléments fondamentaux du gameplay. La classe Game est ainsi composée d'un unique vélo (Bike), d'un timer dédié et d'une collection d'obstacles. Cette relation forte implique que ces éléments n'existent qu'au sein de leur conteneur - leur cycle de vie est entièrement géré par Game. Par exemple, quand une partie se termine, tous les obstacles sont automatiquement détruits avec l'instance de Game.

L'agrégation a été retenue pour les composants d'interface utilisateur. Les écrans Menu, About et Scores existent de manière semi-indépendante - Game les utilise sans en avoir l'exclusivité. Cette relation plus souple permet par exemple de réinitialiser l'écran des scores sans affecter l'état principal du jeu. Elle offre également une meilleure isolation pour les tests unitaires.

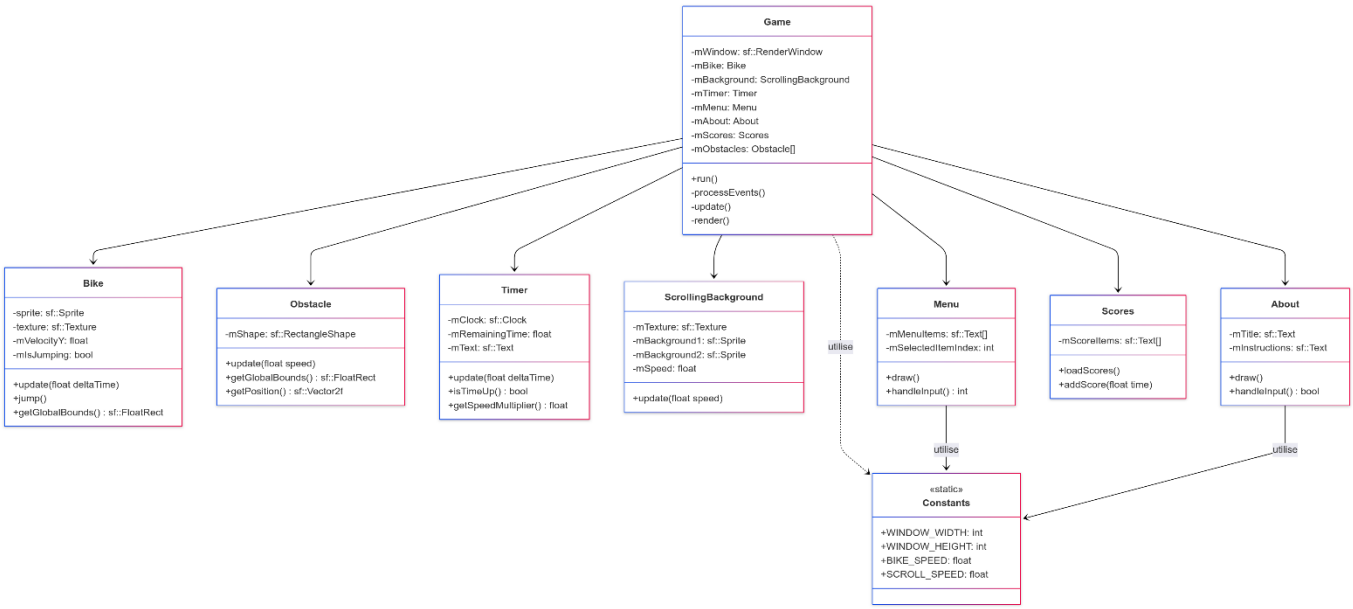


Figure 1 : *Diagramme de classes montrant les relations entre les classes*

La classe Game agit comme le centre névralgique du projet. Elle crée et gère directement plusieurs composants clés : le vélo du joueur (Bike), les obstacles (Obstacle), le chronomètre (Timer), le décor animé (ScrollingBackground), ainsi que les différentes interfaces utilisateur (Menu, About et Scores). Game s'appuie également sur les paramètres globaux définis dans Constants pour maintenir une configuration cohérente à travers tout le jeu.

Les classes Bike et Obstacle interagissent étroitement pour créer le gameplay principal. Bike, contrôlé par le joueur, doit constamment vérifier les collisions avec les Obstacles générés par le système. Ces deux classes sont coordonnées par Game, qui supervise leurs mises à jour et leurs interactions tout au long de la boucle de jeu.

Le Timer joue un rôle crucial dans le rythme du jeu. Non seulement il gère le compte à rebours, mais il influence également la vitesse de défilement du décor (ScrollingBackground) et la fréquence d'apparition des obstacles. Cette relation circulaire permet d'augmenter progressivement la difficulté au fil de la partie.

Les interfaces utilisateur (**Menu, About, Scores**) sont des composants autonomes mais étroitement liés à Game. Elles partagent toutes l'accès aux constantes globales pour maintenir une cohérence visuelle, tout en étant activées ou désactivées par Game selon l'état courant du jeu.

Enfin, **la classe Constants** sert de référence centrale pour l'ensemble du projet. Toutes les autres classes y accèdent pour obtenir des paramètres critiques, ce qui permet des modifications globales rapides et garantit l'uniformité des comportements à travers les différents composants du jeu.

⇨ Flux des États du Jeu :

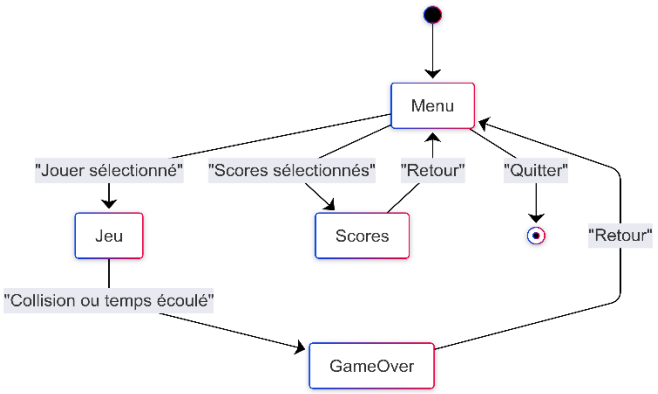


Figure 2 : *Diagramme des états (Gameplay)*

L'expérience joueur commence toujours par l'état **Menu**, l'écran d'accueil qui sert de point de départ à toutes les interactions. Depuis cet état central, trois transitions principales sont possibles, guidant le joueur à travers les différentes fonctionnalités du jeu.

Lorsque le joueur sélectionne "Jouer", le système passe à l'état **Jeu**, le cœur de l'expérience interactive. Dans cet état, toutes les mécaniques de gameplay entrent en action : contrôle du vélo, génération d'obstacles, et gestion du temps. Deux issues possibles terminent cette phase : soit une collision avec un obstacle, soit l'expiration du temps imparti, déclenchant dans les deux cas la transition vers l'état **GameOver**.

L'écran GameOver marque la fin d'une partie. Conçu comme un état transitoire, il présente les résultats tout en offrant un simple retour vers le menu principal. Cette boucle intentionnelle (Menu → Jeu → GameOver → Menu) crée un cycle de jeu satisfaisant qui encourage à rejouer.

La consultation des scores forme une boucle secondaire accessible depuis le menu.
L'état **Scores** affiche les performances passées dans un format clair, avec la même option de retour au menu que GameOver. Cette symétrie dans les transitions maintient une navigation intuitive.

L'option "Quitter" agit comme transition terminale, fermant proprement l'application. Positionnée stratégiquement dans le menu principal, elle permet une sortie immédiate sans étapes intermédiaires, tout en étant suffisamment discrète pour ne pas interrompre accidentellement l'expérience de jeu.

2.3. Avantages d’architecture de projet :

L’architecture modulaire adoptée pour ce projet de jeu de course à vélo offre plusieurs avantages majeurs qui renforcent la qualité globale de l’application. La séparation claire des composants dans une structure de dossiers bien organisée permet une maintenance simplifiée et un développement plus efficace. Chaque élément du jeu possède sa propre classe spécialisée, ce qui suit le principe de responsabilité unique et facilite les modifications ciblées sans affecter l’ensemble du système.

L’architecture centrale repose sur la classe **Game** qui agit comme contrôleur principal, coordonnant harmonieusement les différents modules tout en gérant la boucle de jeu essentielle. Cette approche permet une meilleure gestion des états du jeu (menu, partie, scores) et des transitions fluides entre eux. Les classes **Bike** et **Obstacle**, conçues pour une interaction optimale, implémentent des mécaniques de gameplay précises tout en maintenant des performances stables grâce à leur conception orientée objet rigoureuse.

Le système de **Timer** introduit une dimension stratégique au gameplay, tandis que le **ScrollingBackground** enrichit l’expérience visuelle par des techniques professionnelles de rendu

graphique. Les interfaces utilisateur (**Menu**, **About**, **Scores**) bénéficient d'une conception cohérente et intuitive, séparant nettement la logique d'affichage des autres composants. Enfin, la classe **Constants** centralise tous les paramètres configurables, offrant un point de contrôle unique pour l'équilibrage et l'adaptation du jeu.

Cette structure bien équilibrée présente plusieurs forces : une extensibilité aisée pour l'ajout de nouvelles fonctionnalités, une testabilité améliorée grâce à l'isolation des composants, et des performances optimisées par une gestion efficace des ressources. La séparation entre code (src), déclarations (include) et assets suit les meilleures pratiques du développement logiciel moderne, tout en permettant une collaboration efficace entre développeurs travaillant sur différentes parties du projet.

III. Implémentation :

3.1. Interface graphique (SFML) :

L'interface du jeu a été développée en utilisant les capacités graphiques de **SFML**, offrant une expérience utilisateur fluide et intuitive. Le **menu principal** présente trois boutons cliquables : *Jouer* pour lancer une partie, *Scores* pour consulter les meilleurs résultats, et *Quitter* pour fermer l'application. Ces boutons réagissent aux interactions souris/clavier avec des effets visuels (changement de couleur, animations).

Durant le **gameplay**, l'immersion est renforcée par un **arrière-plan défilant** (*ScrollingBackground*) qui crée une illusion de mouvement. Les éléments visuels (vélo, obstacles) sont affichés via des sprites optimisés, tandis que la police *Anton-Regular.ttf* assure une excellente lisibilité des textes (chronomètre, scores). L'ensemble maintient un style graphique cohérent sur tous les écrans.

a. **Menu Principal :**

Capture : Menu.jpg

L'écran d'accueil présente une interface épurée avec quatre options claires :

- Titre du jeu en en-tête
- Boutons cliquables centrés : Jouer, Scores, À Propos, Quitter
- Style minimaliste avec police lisible (Anton-Regular)
- Effets interactifs au survol (changement de couleur)

Exemple d'interaction : Sélection de « Jouer » lance immédiatement la partie.

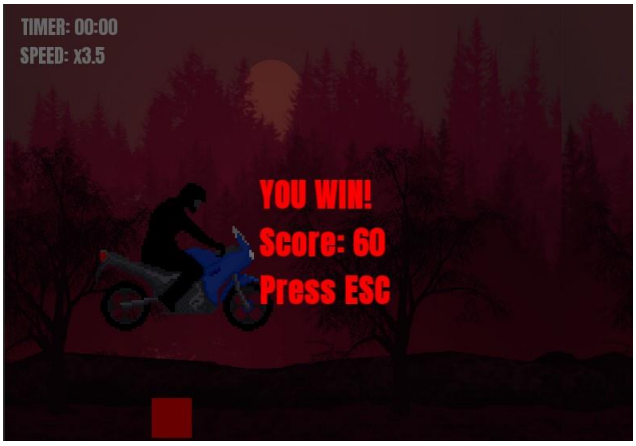
b. Écran de Jeu :

Capture : Jeu.jpg

Interface pendant la partie :

- HUD informatif : Chronomètre (00 :48) + indicateur de vitesse (x1.5)
- Vélo positionné au centre de l'écran
- Obstacles mobiles à droite (génération procédurale)
- Décor dynamique avec effet de défilement fluide

c. Écran de Victoire :



Capture : Win.jpg

Affiché quand le temps expire (00 :00) :

- Message de réussite (« YOU WIN ! ») en évidence
- Statistiques : Score maximal (60), vitesse finale (x3.5)
- Option de sortie (« Press Est ») pour retour au menu.

d. Écran Game Over :

Capture : Gameover.jpg

Déclenché par collision ou temps insuffisant :

- Alerte « GAME OVER » en rouge (effet visuel impactant)
- Résumé de performance : Temps (00 :50), score (9), vitesse (x1.4)
- Prompt clair (« Appuyez sur Echap ») pour rejouer

3.2. Mécanismes du jeu (défilement, collision, score) :

Le **défilement parallaxe** de l'arrière-plan est géré dynamiquement : le décor se déplace en continu et se réinitialise hors-champ pour un bouclage infini. Ce mécanisme est synchronisé avec la vitesse du vélo pour une expérience cohérente.

Les **collisions** entre le vélo et les obstacles sont détectées via une méthode de bounding-box (*sf ::FloatRect*), qui compare les positions et dimensions des sprites. En cas de collision, la partie se termine immédiatement (*gameOver*).

Le **système de score** évalue la performance en fonction du temps restant. Les résultats sont sauvegardés dans un fichier texte local, permettant une persistance entre les sessions. Le classement est trié automatiquement pour mettre en valeur les meilleures performances.

3.3. Défis rencontrés et solutions :

Plusieurs défis ont émergé lors de l'implémentation :

- **Sauts irréalistes** : Initialement, les mouvements du vélo manquaient de naturel. L'ajout d'une **gravité** (*velocityY += GRAVITY*) et d'une force de saut ajustable a résolu le problème.
- **Collisions imprécises** : Les premières versions généraient des faux positifs. Le passage à *sf ::FloatRect* pour les calculs de collision a amélioré la précision.
- **Fluidité inégale** : Des ralentissements étaient perceptibles sur certains PCs. La limitation volontaire à **60 FPS** (*setFramerateLimit*) a stabilisé les performances sans sacrifier le confort de jeu.

IV. Résultats et tests :

4.1. Fonctionnalités Réalisées :

Le projet a abouti à la mise en œuvre complète des fonctionnalités clés initialement prévues. Le menu interactif offre une navigation intuitive entre les différentes options (Jouer, Scores, Quitter), avec des effets visuels lors de la sélection. Les contrôles du vélo répondent de manière fluide et précise : les touches directionnelles gèrent les déplacements latéraux, tandis que la barre d'espace permet d'effectuer des sauts simples ou doubles, selon le timing d'exécution.

La génération procédurale des obstacles a été implémentée avec succès, créant un défi dynamique et renouvelé à chaque partie. Les obstacles apparaissent à intervalles réguliers tout en accélérant progressivement, augmentant la difficulté de manière équilibrée. Enfin, le système de score persistant enregistre fidèlement les performances des joueurs dans un fichier externe, conservant ainsi l'historique des parties même après la fermeture de l'application.

4.2. Tests et Validation :

Une batterie de tests rigoureux a été menée pour valider le bon fonctionnement du jeu. Le système de saut a été vérifié sous différents angles : pression courte/longue, enchaînement rapide, et combinaison avec les déplacements. Dans tous les cas, le vélo réagit comme prévu, avec une physique réaliste (gravité, inertie).

Les collisions ont fait l'objet d'une attention particulière. Des tests intensifs ont confirmé que l'impact avec un obstacle déclenche immédiatement l'écran de fin de partie, sans faux positifs ni retard. De même, le chronomètre a été validé : lorsqu'il atteint zéro, la victoire est correctement enregistrée et le score final sauvegardé. Ces tests ont été répétés sur différentes configurations matérielles pour garantir une expérience uniforme.

V. Conclusion et perspectives :

Ce projet nous a permis d’appliquer concrètement la POO en C++ et de découvrir la bibliothèque SFML pour le développement de jeux 2D. Nous avons appris à concevoir plusieurs classes coopérant dans un jeu, à gérer la boucle de rendu, les entrées clavier et les collisions. Ce travail a renforcé notre compréhension de l’encapsulation (chaque classe gère ses propres données), de la composition (Game contient Bike, Obstacles, etc.) et nous a fait pratiquer la documentation et l’interface graphique avec SFML.

En termes d’améliorations, on pourrait ajouter des animations plus sophistiquées, un système de niveaux croissants, des effets sonores (utilisant le module audio de SFML), ou un classement des meilleurs scores. On pourrait aussi optimiser le design des classes (par exemple introduire une classe de base pour les écrans Menu/About) et mieux séparer la logique de jeu de la partie graphique. Enfin, l’ergonomie du jeu (menus plus détaillés, tutoriel) et la gestion des ressources (textures, sons) pourraient être améliorées.

VI. Annexes :

- Références bibliographiques :

1. **Documentation Officielle SFML**
    - *Titre* : SFML 2.6 Documentation
    - *Lien* : https://www.sfml-dev.org/documentation/2.6.1/

- *Utilité* : Référence complète pour les classes SFML (RenderWindow, Sprite, Text) utilisées dans le projet.

2. **Programmation Orientée Objet**
   - *Ouvrage* : "Design Patterns: Elements of Reusable Object-Oriented Software" (Gamma et al.)
   - *Édition* : Addison-Wesley Professional (1994)
   - *Utilité* : Base théorique pour les principes d'encapsulation, composition et héritage appliqués.
3. **Tutoriels C++ Modernes :**
   - *Site* : Learn C++ (https://www.learncpp.com/)
   - *Chapitres pertinents* :
     - Gestion mémoire (smart pointers)
     - Surcharge d'opérateurs
   - *Utilité* : Bonnes pratiques pour la structure du code.
4. **Optimisation de Jeux 2D :**
   - *Article* : "2D Game Collision Detection" (Mozilla Developer Network)
   - *Lien* : https://developer.mozilla.org/...
   - *Utilité* : Algorithmes pour les collisions via *sf::FloatRect*.
5. **Gestion de Projet :**
   - *Outil* : Documentation CMake (https://cmake.org/documentation/)
   - *Utilité* : Configuration du système de build multiplateforme.
6. **Ressource assets :**
   - *Textures :*
     - ✓ Background **:** https://itch.io/
     - ✓ Moto : https://opengameart.org/
   - *Fonts :*
     - ✓ *https://fonts.google.com/*