ⵜⴰⵖⵍⵓⵍⵜ ⵏ ⵜⵙⴻⵎⵎⵓⵔⵉ
FACULTÉ DES SCIENCES
MARRAKECH
UNIVERSITÉ CADI AYYAD
كلية العلوم
FACULTY OF SCIENCE
Semlalia

Youssef EL RHOMARI
Mohamed BALJID
IAP-S4
2025-2026

Développement d'un mini-jeu de course en C++ avec gestion d'obstacles et d'un trajet

1. Introduction :

Ce projet s'inscrit dans le cadre du module de programmation orientée objet en C++. Il a pour objectif la conception et la réalisation d'un mini-jeu de course, dans lequel le joueur contrôle une voiture devant éviter les obstacles sur la route tout en progressant sur un trajet.

Ce projet nous permet de mettre en pratique les notions de programmation orientée objet, la gestion d'événements en temps réel, l'utilisation de bibliothèques graphiques (comme SFML), ainsi que la structuration d'un projet logiciel en C++.

À travers ce rapport, nous allons détailler les étapes de conception, le choix des outils, l'implémentation du jeu ainsi que les difficultés rencontrées durant le développement.

2. Conception et modélisation :

La conception du jeu a été structurée de manière modulaire afin de favoriser la clarté, la réutilisabilité et la maintenabilité du code. Cinq composants principaux ont été identifiés et modélisés sous forme de classes distinctes : **Voiture**, **Obstacle**, **Game**, **Background** et **UI** (Interface Utilisateur). Cette architecture orientée objet permet une séparation claire des responsabilités et facilite l'évolution du projet.

a) Composants du jeu :

- **Voiture :** Représente le véhicule contrôlé par le joueur. Elle peut se déplacer latéralement (haut/bas) en réponse aux entrées clavier. Elle est également responsable de la détection de collisions avec les obstacles.
- **Obstacle :** Objets dynamiques générés périodiquement sur la route. et descendent verticalement pour simuler le mouvement. Le joueur doit les éviter.
- **Game :** Constitue le cœur de l'application. Cette classe centralise la logique du jeu, notamment la boucle principale, la gestion des événements (entrées clavier, fermetures de fenêtre), le calcul des collisions, le score et les conditions de fin de partie.
- **Background :** Gère le fond visuel du jeu, notamment l'effet de défilement vertical de la route et du décor afin de donner l'illusion de vitesse et de déplacement continu.
- **UI (Interface Utilisateur) :** S'occupe de l'affichage des informations essentielles à l'écran : score du joueur, messages contextuels (ex. : "Game Over", "Appuyez sur jouer pour commencer", etc.).

b) Modélisation objet (structure des classes) :

- **Game :** Classe centrale qui coordonne l'ensemble des éléments du jeu. Elle gère les différentes phases (menu, jeu, game over, etc.), orchestre les interactions entre les objets, et contrôle le déroulement général de la partie.
  - ✓ **Responsabilités :**

- **Initialiser la fenêtre et les ressources.**
- **Gérer la boucle principale du jeu.**
- **Mettre à jour et afficher les composants du jeu.**
- **Suivre l'état du jeu et la difficulté.**
- **Détecter les collisions et gérer le score.**

✓ **Principaux attributs :**

- **Une fenêtre de rendu.**
- **Un état du jeu (GameState) et un niveau de difficulté (Difficulty).**
- **Un objet Voiture (le joueur).**
- **Un objet Background.**
- **Une liste d'objets Obstacle.**
- **Une interface utilisateur (UI).**
- **Des variables pour la gestion du temps, des collisions, et du score.**

✓ **Méthodes principales :**

- **run() : lance la boucle du jeu.**
- **update() : met à jour l'état des objets.**
- **render() : affiche les éléments à l'écran.**
- **resetGameForPlay() : réinitialise les variables pour une nouvelle partie.**
- **checkCollisions() : détecte les collisions entre le joueur et les obstacles.**

• **Voiture :** Représente le joueur dans le jeu. Cette classe gère l'apparence et les déplacements latéraux de la voiture contrôlée par le joueur.

✓ **Responsabilités :**

- **Charger et afficher la texture de la voiture.**
- **Positionner correctement la voiture sur la route.**
- **Gérer le déplacement du joueur (haut/bas).**
- **Fournir l'accès au sprite pour la détection de collision.**

✓ **Principaux attributs :**

- **texture : image utilisée pour représenter la voiture.**
- **sprite : entité graphique affichée à l'écran.**

✓ **Méthodes principales :**

- **initTexture() : charge l'image depuis un fichier.**
- **initSprite() : initialise la position de départ et adapte la taille.**
- **update() : met à jour la position de la voiture selon les entrées clavier.**
- **render() : dessine la voiture à l'écran.**
- **getSprite() : retourne une référence au sprite pour les collisions.**

• **Obstacle :** Représente un obstacle que le joueur doit éviter. Les obstacles apparaissent dynamiquement sur la route et se déplacent vers la gauche pour simuler le mouvement.

✓ **Responsabilités :**

- **Choisir aléatoirement une texture parmi celles disponibles.**
- **Positionner un obstacle sur la route.**
- **Gérer son déplacement horizontal.**
- **Fournir les informations nécessaires pour la détection de collision.**
- **Indiquer quand l'obstacle sort de l'écran (pour le supprimer).**

✓ **Principaux attributs :**

- **sprite : entité graphique représentant l'obstacle (flaque, voiture, trou, etc.).**

✓ **Méthodes principales :**

- **update() : met à jour la position de l'obstacle selon la vitesse du jeu.**
- **render() : dessine l'obstacle à l'écran.**
- **getSprite() : retourne le sprite pour les collisions.**
- **isOffScreenLeft() : vérifie si l'obstacle est sorti de l'écran par la gauche.**

- **Background :** Gère l'arrière-plan visuel du jeu, notamment la route et la ville lointaine. Crée un effet de défilement horizontal pour simuler le mouvement de la voiture.

✓ **Responsabilités :**

- **Charger et afficher les textures de fond.**
- **Créer un effet de déplacement fluide (scrolling).**
- **Adapter la taille et la position des éléments au format de la fenêtre.**
- **Fournir des informations sur la position de la route pour placer les obstacles et la voiture.**

✓ **Principaux attributs :**

- **roadTexture, farCityTexture : textures utilisées pour la route et l'arrière-plan.**
- **roadSprite1, roadSprite2 : deux sprites utilisés pour créer un défilement continu de la route.**
- **farCitySprite1, farCitySprite2 : sprites de l'arrière-plan lointain.**
- **roadScaleY : facteur d'échelle vertical utilisé pour adapter la route à la fenêtre.**

✓ **Méthodes principales :**

- **initTextures() : charge les textures à partir des fichiers.**
- **initSprites() : initialise les sprites selon la taille de la fenêtre.**
- **update() : fait défiler les sprites vers la gauche.**
- **render() : dessine les sprites du fond à l'écran.**
- **getRoadVisualHeight() : retourne la hauteur visible de la route (utile pour les limites de déplacement).**
- **getRoadTopY() : retourne la position Y du haut de la route.**

- **UI (User Interface) :** Gère l'affichage et l'interaction avec l'utilisateur dans les différents états du jeu (menu, paramètres, jeu, fin de partie, confirmation de sortie).

✓ **Responsabilités :**

- **Afficher tous les textes liés à l'interface utilisateur.**
- **Gérer les clics et le survol sur les boutons de texte.**
- **Permettre la navigation entre les états du jeu.**
- **Afficher et mettre à jour dynamiquement des informations (score, volume, difficulté…).**

✓ **Principaux attributs :**

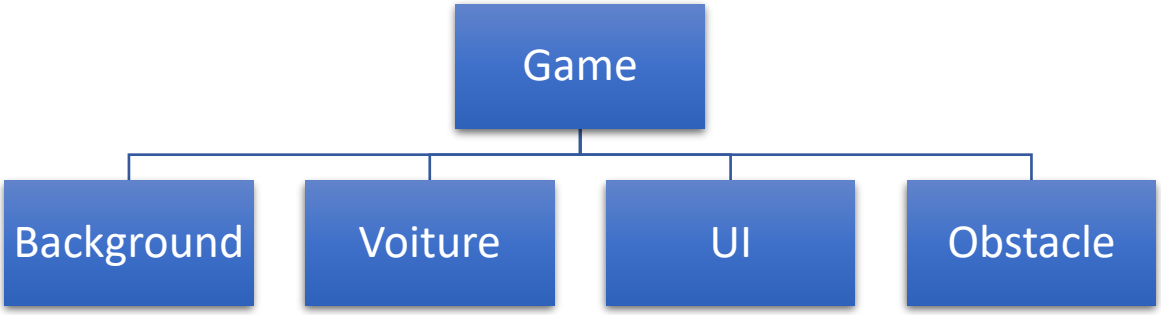
- **sf::Font uiFont : police utilisée pour tous les textes.**
- **Plusieurs objets sf::Text :**
  - **Pour le menu principal : titre, boutons Jouer / Paramètres / Quitter.**
  - **Pour les paramètres : difficulté, volume, high score, boutons de réglage.**
  - **Pour l'écran Game Over : score final, high score, bouton retour.**
  - **Pour la confirmation de sortie : texte de confirmation, boutons Oui / Non.**

✓ **Méthodes principales :**

- **loadFont() : charge une police depuis un fichier.**
- **setupElements() : initialise et positionne tous les éléments de texte.**
- **setupText() : méthode utilitaire pour configurer un objet sf::Text.**
- **processEvent() : gère les clics de souris et déclenche les actions correspondantes.**
- **update() : met à jour les effets visuels (ex. : changement de couleur au survol).**
- **render() : affiche les textes correspondant à l'état courant du jeu.**
- **updateHighScoreDisplayTexts() : met à jour l'affichage du high score.**
- **Méthodes utilitaires privées : isMouseOverText() et updateButtonStates().**

c) Schéma de conception :

Youssef EL RHOMARI
Mohamed BALJID
IAP-S4



3. Environnement de travail :

Le développement du mini-jeu a été réalisé dans un environnement adapté à la programmation C++ orientée objet, avec une bibliothèque graphique pour gérer l'affichage et les événements.

a) Outils utilisés :

- **IDE : Visual Studio**
  L'IDE Visual Studio est un panneau de lancement créatif que vous pouvez utiliser pour modifier, déboguer et générer du code, puis publier une application. En plus de l'éditeur et du débogueur standard fournis par la plupart des IDE, Visual Studio inclut des compilateurs, des outils de complétion de code, des concepteurs graphiques et bien d'autres fonctionnalités pour améliorer le processus du développement de logiciels. IDE le plus complet pour les développeurs .NET et C++ sur Windows. Entièrement rempli d'un bon ensemble d'outils et de fonctionnalités permettant d'élever et d'améliorer chaque étape du développement de logiciels.
- **Compilateur** : MSVC (Microsoft Visual C++)
  Intégré directement à Visual Studio pour compiler les fichiers C++.
- **Bibliothèque graphique** : **SFML (Simple and Fast Multimedia Library)**
  SFML est une interface de programmation bas niveau destinée à construire des jeux de vidéo ou des programmes interactifs. Elle est écrite en C++, mais également disponible dans divers langages
  comme C, D, Python, Ruby ou Microsoft .NET. Elle a entre autres pour but de proposer une alternative orientée objet à la SDL.Elle a également la particularité de fournir un graphisme 2D accéléré en utilisant OpenGL en interne, qui permet à l'utilisateur de s'affranchir de la gestion d'une pseudo-3D.Composée de nombreux modules, elle peut être utilisée en tant que système de fenêtrage minimal pour s'interfacer avec OpenGL, ou en tant que

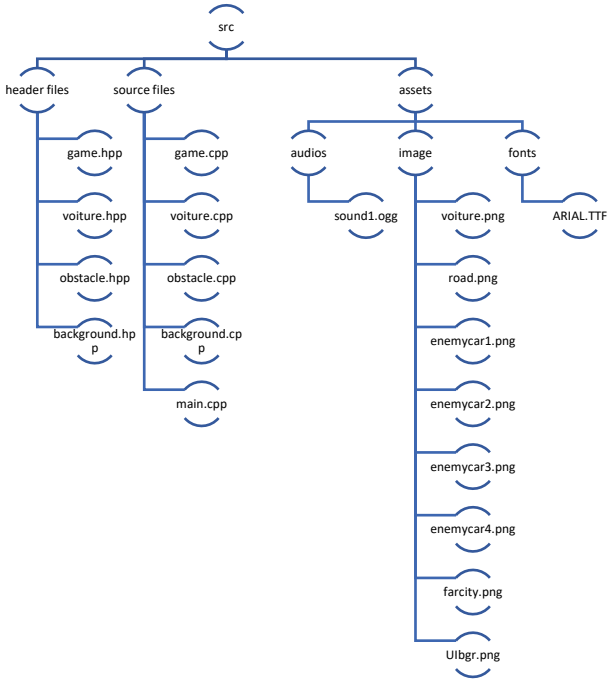
bibliothèque multimédia riche en fonctionnalités pour construire des jeux vidéo ou des programmes interactifs.

Employée pour :

- La gestion de la fenêtre du jeu
- L'affichage des textures (voiture, obstacles, route…)
- Le traitement des événements clavier
- L'affichage du texte (score, temps, messages)

b) Organisation des fichiers :

- **L'organisation de ce jeu est sous la forme suivant :**



4. Implémentation :

Le projet a été développé en suivant une architecture orientée objet, avec des classes bien séparées pour chaque composant du jeu. Voici un aperçu des principaux éléments implémentés, accompagnés de leurs rôles dans le jeu.

a) Classe Game

C'est la classe principale qui gère :

- La création de la fenêtre du jeu
- La boucle principale (main loop)
- Le rafraîchissement de l'écran
- La gestion des événements clavier
- L'appel aux fonctions de mise à jour et d'affichage des autres classes (Voiture, Obstacle, Background, UI)

b) Classe Voiture

- Initialise la position et la texture de la voiture
- Gère les déplacements haut/bas à l'aide des touches fléchées
- Contient une méthode de détection de collision avec les obstacles

c) Classe Obstacle

- Génère des obstacles à des positions aléatoires.
- Gère leur déplacement vers le bas
- Supprime les obstacles une fois hors de l'écran

d) Classe Background

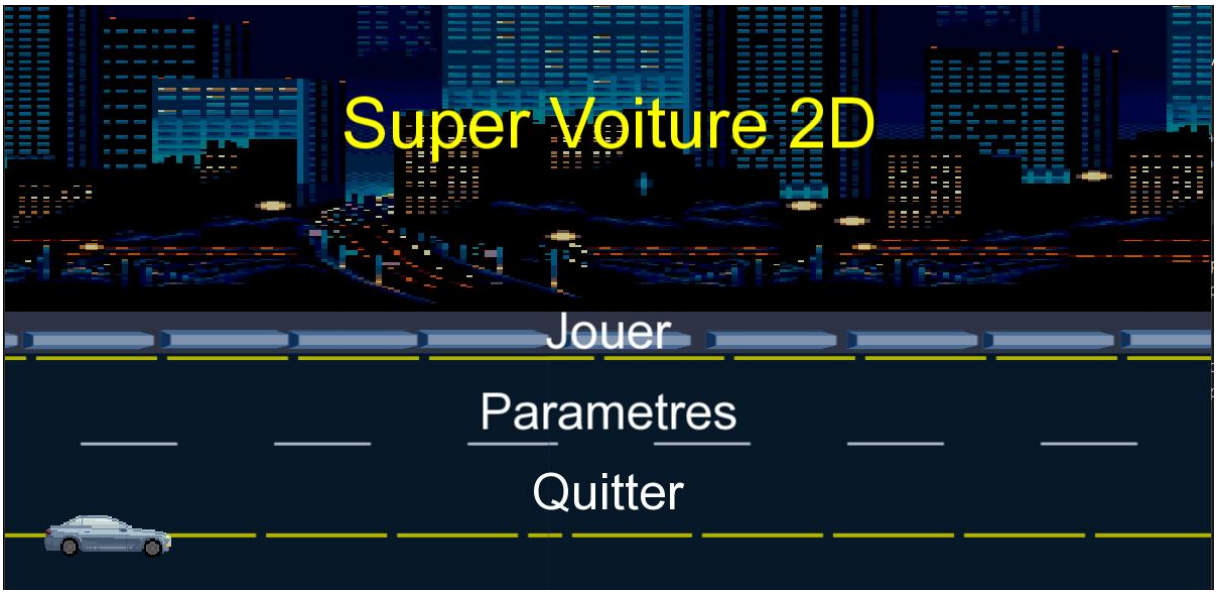
- Crée un effet de défilement vertical en boucle pour simuler un mouvement constant de la route
- Améliore l'immersion du joueur

e) Classe UI

- Affiche :
  - Le score (basé sur le temps ou la distance parcourue)
  - Le chronomètre
  - Les messages importants (ex. : "Game Over", "jouer")

f) Captures d'écran

- Interface de démarrage



Youssef EL RHOMARI
Mohamed BALJID
IAP-S4

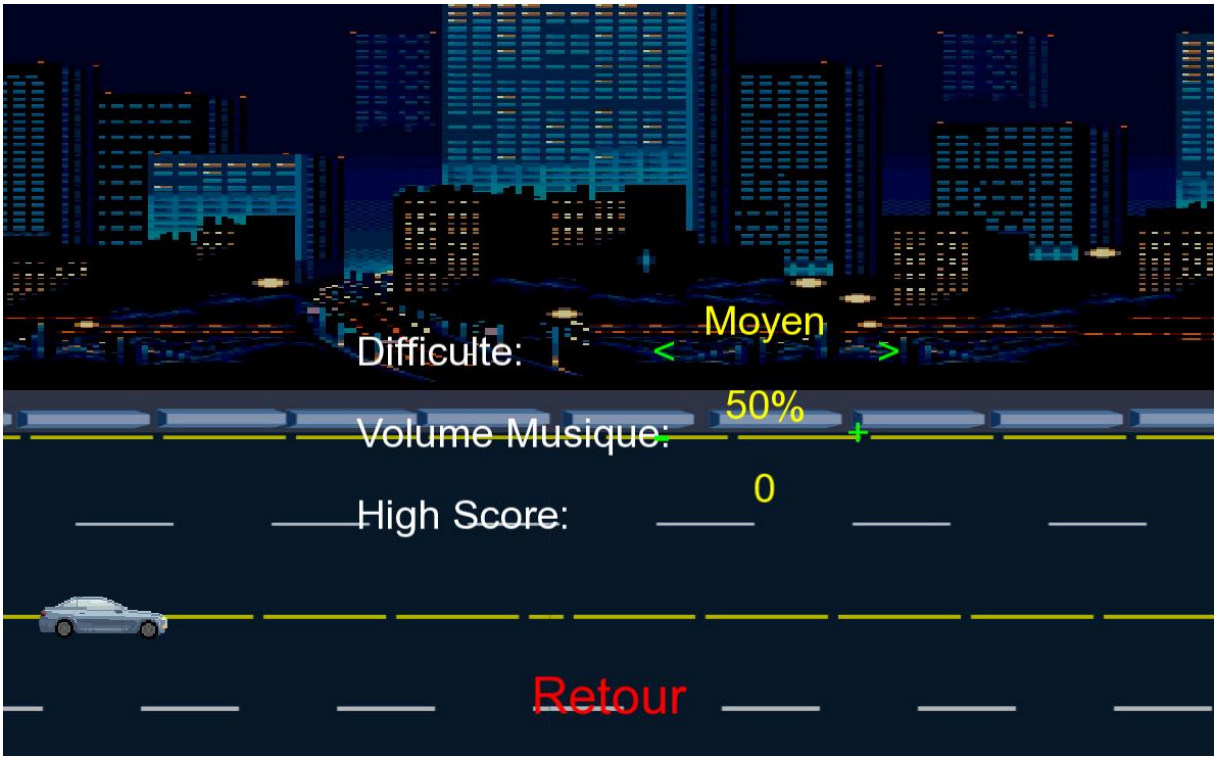
- Voiture en mouvement

- Collision avec un obstacle

- Affichage du score ou du chrono

Youssef EL RHOMARI
Mohamed BALJID
IAP-S4

- Interface de paramètre :



g) Problèmes rencontrés

- Pendant le développement du jeu, plusieurs problèmes techniques ont été rencontrés, notamment liés à l'affichage, à la gestion des collisions, et au positionnement des éléments :

- **Défilement de la route (background)** : Il était difficile au début de créer un effet fluide de défilement continu. La solution adoptée a été de dupliquer l'image du fond et de faire défiler deux instances l'une après l'autre en boucle, créant ainsi un effet sans coupure visuelle.
- **Positionnement initial de la voiture** : Trouver une position cohérente au centre de la route a demandé plusieurs ajustements. J'ai utilisé les dimensions de la fenêtre et de la texture pour centrer la voiture horizontalement et la placer juste au-dessus du bas de l'écran.
- **Positionnement des obstacles** : Les obstacles apparaissaient parfois en dehors de la route ou se superposaient. Pour corriger cela, j'ai défini une plage horizontale spécifique (en fonction des bords de la route) et introduit un espacement minimum entre deux obstacles générés.
- **Gestion des collisions** : Les collisions étaient soit trop sensibles, soit pas détectées. J'ai opté pour un ajustement manuel des zones de collision en utilisant des rectangles plus petits que les sprites réels, ce qui a amélioré la jouabilité.
- **Affichage du texte (UI)** : L'intégration d'une police personnalisée SFML ne fonctionnait pas au début. J'ai résolu cela en chargeant la police depuis un chemin relatif et en vérifiant que le fichier était bien placé dans le répertoire d'exécution.
- **Performances** : Lorsqu'un trop grand nombre d'obstacles étaient en mémoire, des ralentissements pouvaient survenir. J'ai mis en place un système de suppression automatique des obstacles sortis de l'écran pour optimiser les performances.

5. Conclusion :

Ce mini-projet de jeu en C++ a été une excellente opportunité pour mettre en pratique les concepts de programmation orientée objet, de gestion graphique avec la bibliothèque SFML, ainsi que l'organisation d'un projet modulaire.

En développant ce jeu de voiture avec obstacles, nous avons pu :

- Appliquer la structure en classes pour organiser clairement les différentes fonctionnalités (logique de jeu, affichage, entrées utilisateur…),
- Manipuler des éléments graphiques, tels que les sprites, le texte et le fond défilant,
- Gérer des interactions complexes, comme les collisions et le chronomètre,
- Résoudre des problèmes concrets de développement liés à l'affichage, au positionnement et aux performances.

Malgré les défis rencontrés, le projet a été mené à terme avec succès, offrant une expérience de jeu simple mais fonctionnelle et fluide.

Ce travail a renforcé notre compréhension de la programmation en C++ et nous a permis d'explorer le domaine du développement de jeux en 2D, ce qui peut servir de base pour des projets plus avancés à l'avenir.

6. Références :

- Pour apprendre les nouvelles notions : https://www.sfml-dev.org/fr/tutorials/3.0/
- Pour les images : https://fr.freepik.com/