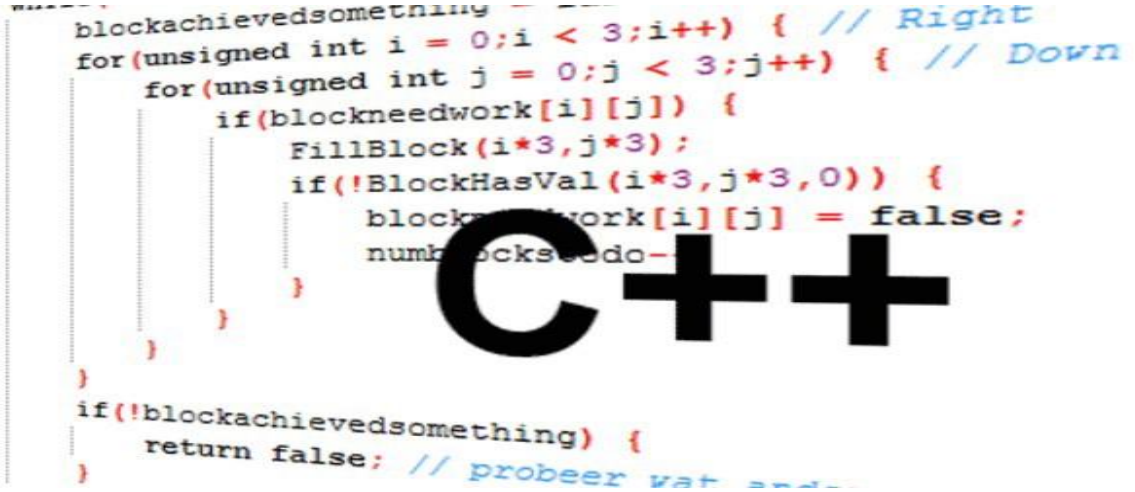
MINI PROJET « PROGRAMMATION EN C++ »



TITRE :

jeu de course à velo

Réalisé à :

Faculté des sciences semlalia

Par :

- **karima boutskaouin**
- **Hanane ait bah**

Encadré par :

prof hannane rachida

Table des matières

I Introduction 3
II Installation de raylib sur Linux (Ubuntu) 4
II.1 Etapes d'installation de raylib sous linux 4
II.2 Choix de l'Environnement de Développement 6
II.2.1 objectifs pédagogiques et techniques du choix 6
II.2.2 Gedit : éditeur simple mais puissant 6
III Outils Utilisés 7
III.1 Git et GitHub 7
III.1.1 Git : Un Outil de Contrôle de Version 7
III.1.2 GitHub : Collaboration à Distance 8
IV Conception et Modélisation du Projet 8
IV.1 Structure du Projet 8
IV.2 Principe du jeu 11
IV.3 Architecture du jeu 11
V conclusion : 14

I Introduction

*Bienvenue dans "**Jeu du Cycliste**", un jeu 2D développé en **C++** avec la bibliothèque **raylib**, où le joueur incarne une personne à vélo qui doit **éviter des obstacles** sur son chemin. Le jeu repose sur des mécaniques simples mais addictives, basées sur la **réactivité**, **l'anticipation**, et **le timing**.*

Ce projet a été conçu dans le but de se familiariser avec la programmation de jeux en C++, tout en explorant les possibilités offertes par **raylib**, une bibliothèque graphique moderne, légère et simple à prendre en main. Le jeu met en scène un environnement dynamique dans lequel le joueur doit manœuvrer son vélo pour éviter des obstacles aléatoires qui apparaissent progressivement à l'écran.

Le joueur devra ainsi faire preuve de réflexes rapides pour éviter des objets tels que des rochers, des plots de signalisation, ou d'autres dangers, tout en parcourant la plus grande distance possible. Le score est basé sur la distance parcourue sans collision, ce qui pousse le joueur à améliorer sa performance à chaque essai.

Le choix de raylib s'est imposé pour plusieurs raisons :

- Une API claire et bien documentée
- Une prise en charge facile des fenêtres, du rendu 2D, des entrées clavier/souris
- Une compilation rapide et multiplateforme (Windows, Linux, macOS)

Le développement de ce jeu a permis de mettre en pratique plusieurs compétences en programmation :

- Utilisation de boucles de jeu en temps réel
- Gestion des entrées clavier
- Détection de collisions entre objets
- Création et animation de sprites
- Organisation modulaire du code C++ pour séparer la logique du jeu, le rendu, et la gestion des événements

Le choix de **raylib** pour le développement de ce jeu n'est pas anodin. Plusieurs raisons techniques et pédagogiques justifient son utilisation :

➢ Simplicité d'utilisation :

Raylib a été conçu pour être **simple, claire et intuitive**, notamment pour les débutants ou les développeurs souhaitant créer rapidement des prototypes. Son API suit une logique cohérente avec des fonctions comme `DrawRectangle()`, `IsKeyDown()`, `LoadTexture()`, etc., qui sont très lisibles.

➢ Idéale pour le C et C++

Contrairement à des moteurs lourds comme Unity ou Unreal, raylib est une **bibliothèque légère et écrite en C**, parfaitement adaptée à la programmation en **C++** sans surcouche inutile. Cela permet un meilleur contrôle du code et une meilleure compréhension des bases de la programmation graphique.

➢ Documentation claire et communauté active

Raylib dispose d'une documentation très bien faite ainsi qu'une communauté active. Cela facilite grandement l'apprentissage, la recherche de solutions, et le partage de projets.

➢ Support natif du 2D

Le moteur est spécialement adapté pour le développement de **jeux 2D**, avec de nombreuses fonctions intégrées pour le dessin de formes, le rendu d'images, les animations, la gestion de clavier/souris, le son, etc.

➢ Multiplateforme

Raylib fonctionne sur Linux, Windows, macOS, et même sur des plateformes embarquées. Cela garantit une portabilité maximale du projet, un avantage important pour tout développeur indépendant ou étudiant.

II Installation de raylib sur Linux (Ubuntu)

II.1 Etapes d'installation de raylib sous linux

- **Pour installer raylib sous Linux, voici les étapes recommandées :**

✓ Installer les dépendances nécessaires :

Avant d'installer raylib, assure-toi que les outils de compilation sont présents :

```
sudo apt update
sudo apt install build-essential cmake git libasound2-dev libpulse-dev libx11-
dev libxcursor-dev libxrandr-dev libxinerama-dev libxi-dev libgl1-mesa-dev
libegl1-mesa-dev
```

✓ Cloner le dépôt officiel de raylib

```
git clone https://github.com/raysan5/raylib.git
cd raylib
```

✓ Compiler et installer raylib

```
mkdir build && cd build
cmake ..
make -j$(nproc)
sudo make install
```

- ✓ Vérification de l'installation Crée un fichier test main.cpp :

```
#include "raylib.h"

int main() {

InitWindow(800, 600, "Test Raylib"); while
(!WindowShouldClose()) {
        BeginDrawing();
        ClearBackground(RAYWHITE);
        DrawText("Raylib fonctionne !", 190, 200, 20, DARKGRAY);
        EndDrawing();
    }
    CloseWindow();
    return 0;
}
```

ompile-le avec :

```
g++ main.cpp -o test -lraylib -lGL -lm -lpthread -ldl -lrt -lX11
```

Puis lance le jeu :

```
./test
```

II.2 Choix de l'Environnement de Développement

Le choix de l'environnement de développement joue un rôle fondamental dans la réussite d'un projet logiciel, en particulier lorsqu'il s'agit d'un projet en C++ orienté graphique, comme ce jeu vidéo développé avec raylib. Dans ce projet, nous avons opté pour un environnement minimaliste, transparent et orienté apprentissage : le terminal Linux en combinaison avec l'éditeur de texte Gedit.
Ce choix peut sembler atypique face à des environnements de développement intégrés (IDE) modernes comme Visual Studio Code, CLion ou QtCreator. Cependant, il s'inscrit dans une volonté claire : maîtriser pleinement le processus de compilation, de gestion des dépendances, et d'exécution d'un programme en C++, sans recours à des outils qui automatisent excessivement ces tâches.

II.2.1 objectifs pédagogiques et techniques du choix

- ✓ Compréhension du fonctionnement bas niveau
  - Compilation manuelle avec `g++`
  - Gestion explicite des bibliothèques (`-lraylib`, `-lGL`, etc.)
  - Exécution contrôlée via la console
  - Navigation dans l'arborescence du projet

- ✓ Légèreté et accessibilité
  - Aucun besoin d'installer un IDE lourd
  - Fonctionne sur n'importe quelle distribution Linux standard
  - Utilisable même sur des machines peu puissantes ou en environnement serveur distant

- ✓ Stimulation de l'autonomie du développeur
  - Habituation à la ligne de commande
  - Capacité à résoudre les erreurs de compilation manuellement
  - Maîtrise des outils GNU/Linux essentiels au développement mode

II.2.2 Gedit : éditeur simple mais puissant

II.2.2.1 Présentation

Gedit est l'éditeur de texte officiel de l'environnement de bureau GNOME. Il s'agit d'un outil graphique, léger, souvent préinstallé sur les distributions Linux telles que Ubuntu, Debian, Fedora

II.2.2.2 Fonctionnalités pertinentes pour le C++ :

- **Coloration syntaxique** pour le C, C++, Python, etc.
- **Numérotation des lignes**, indentation automatique
- Prise en charge des fichiers multiples via des onglets
- Recherche/remplacement avec expressions régulières
- Extensible via des **plugins** (mini-terminal, console de compilation, etc.)

II.2.2.3 Exemple d'utilisation :

Pour ouvrir un fichier source :

```
gedit src/main.cpp &
```

- **Collaboration sur le Projet de Jeu**

Ce document présente le rapport détaillé sur le développement du jeu réalisé en collaboration entre [ItsHaname]() et [karimaboutskaouin](). Le projet est basé sur l'utilisation de Raylib et a été développé en C++. Ce rapport couvre les outils utilisés pour la gestion de version, ainsi que la conception et la modélisation du projet.

III Outils Utilisés

III.1 Git et GitHub

III.1.1 Git : Un Outil de Contrôle de Version

Git est un système de contrôle de version décentralisé qui permet de gérer les différentes versions du code et de collaborer efficacement entre les développeurs. Grâce à Git, nous avons pu suivre l'historique de chaque modification et revenir à des versions antérieures lorsque cela était nécessaire.

- **Suivi des Modifications** : Chaque modification dans le code est enregistrée par un "commit", ce qui nous permet de suivre l'évolution du projet de manière claire et ordonnée.
- **Branches** : Nous avons utilisé des branches pour travailler sur des fonctionnalités spécifiques sans perturber la branche principale du projet (`main`). Cela nous a permis de développer des fonctionnalités indépendamment et de les tester avant de les intégrer à la version stable.

-Branche principale (haname) : Cette branche contient les principales mises à jour et fonctionnalités réalisées par ItsHaname.

-Branche secondaire (karimak_G) : La branche de mon ami karimaboutskaouin où il a travaillé sur des améliorations spécifiques et des ajouts au projet.

- **Fusion de Code (Merge)** : Après avoir terminé une fonctionnalité sur une branche distincte, nous avons fusionné les changements dans la branche principale en utilisant des "pull requests". Cela nous a permis de discuter et de valider les modifications avant de les intégrer.

III.1.2 GitHub : Collaboration à Distance

GitHub est une plateforme qui nous a permis de centraliser le code source du projet et de collaborer à distance. Voici les fonctionnalités clés que nous avons utilisées :

- **Fork et Pull Request** : Mon ami karimaboutskaouin a fait un fork de mon dépôt afin de travailler sur le projet en parallèle. Une fois ses modifications terminées, il a soumis une **pull request** pour que je puisse revoir et intégrer ses modifications.
- **Issues et Discussions** : Nous avons utilisé les "issues" pour discuter des problèmes rencontrés et suivre l'avancement des tâches. Cela nous a permis de structurer notre travail et d'assurer une bonne communication tout au long du développement.
- **Documentation** : GitHub nous a également permis de maintenir une documentation claire et accessible, ce qui est essentiel pour comprendre rapidement les fonctionnalités du projet et les instructions d'installation.

IV Conception et Modélisation du Projet

IV.1 Structure du Projet

La structure du projet a été pensée pour être modulaire, claire et maintenable. Voici l'organisation générale du projet :

```
/src
    /Game.cpp          # Logique principale du jeu, boucles, gestion des
événements.
    /Bike.cpp          # Gestion du vélo (mouvement, interactions, etc.)
    /Person.cpp        # Gestion du personnage (réactions, contrôles, état)
    /Obstacle.cpp      # Gestion des obstacles (apparition, collisions)
    /Menu.cpp          # Gère le menu du jeu (démarrage, pause, options
```

```
     /main.cpp         # Point d'entrée, démarrage du jeu
    /Game.h            # Déclarations des classes et fonctions pour Game
    /Bike.h            # Déclarations des classes et fonctions pour Bike
    /Person.h          # Déclarations des classes et fonctions pour Person
    /Obstacle.h        # Déclarations des classes et fonctions pour Obstacle
    /Menu.h            # Déclarations des classes et fonctions pour Menu
/assets
    - Images et autres ressources utilisées (sprites, icônes, etc.)
```

Diagramme de Classe

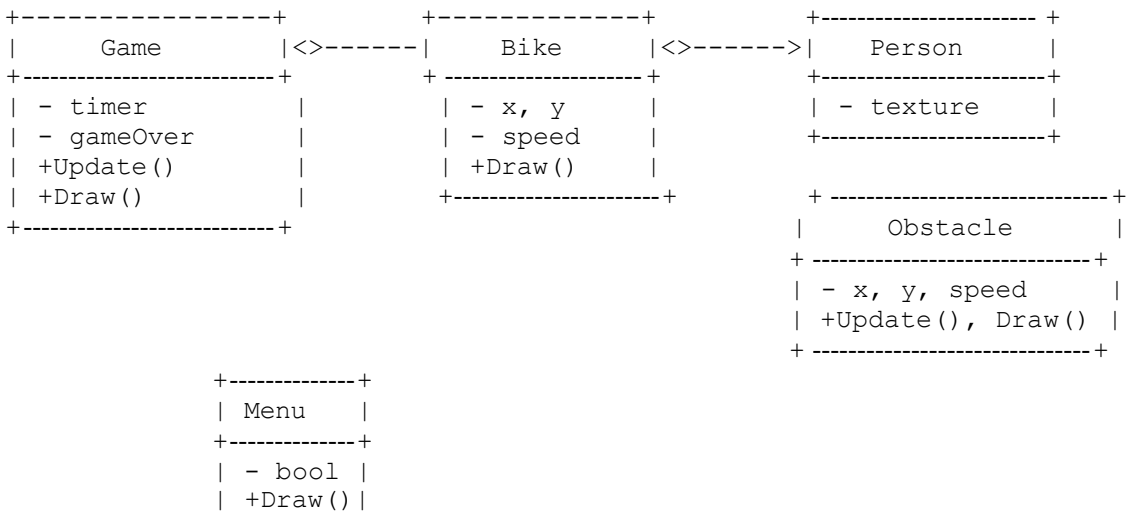
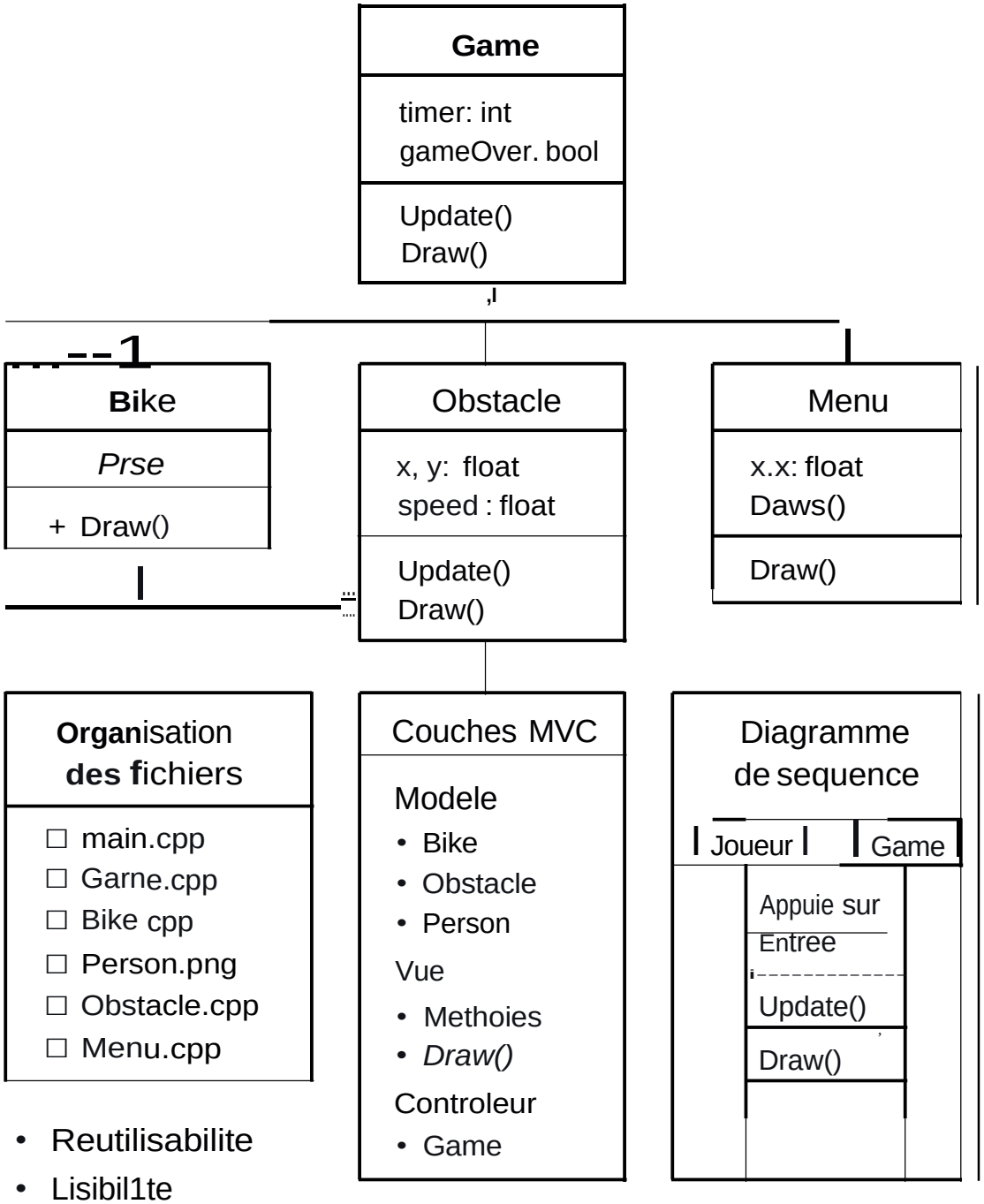


Diagramme de Séquence Simplifié

```
Utilisateur         → App (main)            → Game           → Bike / Obstacle /
Menu
Appuie sur Entree   → InitWindow()          → Update()       → Update()
                    → Boucle de jeu         → Draw()         → Draw()
                    → Quit                  → CloseWindow()
```

Le projet suit une conception orientee objet claire, avec une organisation modulaire et evolutive.



- Reutilisabilite
- Lisibil1te
- Extensibilite
- Maintenance

IV.2 Principe du jeu

Au lancement du programme, un menu principal s'affiche, proposant deux options : Jouer ou Quitter.

- Le joueur peut commencer une partie en appuyant sur **Entrée**.
- Il peut quitter en appuyant sur **Échap** ou **Espace**.

Une fois la partie commencée, un **cycliste sur un vélo** se déplace horizontalement à l'écran. Le joueur doit éviter les obstacles qui apparaissent pendant un temps limité. L'objectif est de survivre jusqu'à la fin du chronomètre

IV.3 Architecture du jeu

La création de ce jeu repose sur une organisation en plusieurs classes, chacune ayant un rôle bien défini.

✓ Classe Velo

Cette classe représente le vélo du joueur, avec une animation dynamique :

- Les roues tournent selon la vitesse du vélo, simulant une rotation fluide.
- Les rayons se déplacent pour renforcer l'illusion de mouvement.
- La position du cycliste (objet de la classe `Person`) change en fonction des touches pressées (haut, bas, gauche, droite).
- L'animation est assurée par la mise à jour continue de la position du vélo et le redessin à chaque frame.

✓ Classe Person

Cette classe représente le personnage sur le vélo. Elle gère son apparence à l'écran.

➢ **Attributs principaux :**

- `width`, `height` : dimensions de la texture.
- `texture` : image représentant le personnage.

➢ **Méthodes :**

- **Constructeur** : charge la texture depuis un fichier image.
- **Destructeur** : libère la mémoire associée à la texture.
- **Draw()** : affiche la texture à une position définie, avec un redimensionnement et un placement adapté à l'écran.

✓ La classe Menu

Le rôle de cette classe est d'afficher un menu qui permet à l'utilisateur de choisir entre démarrer le jeu ou quitter.

-Si l'utilisateur appuie sur Entrée, le jeu commence.

-S'il appuie sur Échap ou Espace, l'application se ferme

✓ la classe obstacle :

Cette classe représente les obstacles que le joueur doit éviter.

- **Fonctionnalités :**
  - Les obstacles ont une forme rectangulaire et se déplacent de droite à gauche à une vitesse constante.
  - Lorsqu'un obstacle sort de l'écran, il réapparaît à droite avec une nouvelle position verticale aléatoire.
- **Composants :**
  - Attributs : position, taille, vitesse.
  - Méthodes :
    - `Update()` : met à jour la position.
    - `Draw()` : dessine l'obstacle.
    - `GetRect()` : retourne un rectangle pour la détection de collision.

✓ Classe Game

Cette classe centralise toute la logique du jeu.

- **Responsabilités :**
  - **Initialisation** : configure le menu, le vélo, les obstacles et le joueur.
  - **Update()** : met à jour les états du jeu (menu, déplacement, collisions, timer...).
  - **Draw()** : affiche les bons éléments selon l'état du jeu (menu, jeu actif, Game Over).
  - **Reset()** : permet de recommencer une nouvelle partie après un Game Over.

- **Fonction main()**

La fonction `main()` est le point d'entrée du jeu.

- **Fonctionnalités :**
  - Initialise la fenêtre avec Raylib.
  - Crée un objet `Game`.
  - Lance une boucle principale :
    - Appelle `Update()` pour la logique.
    - Appelle `Draw()` pour l'affichage.
  - Ferme proprement la fenêtre à la fin du programme

V conclusion :

Ce projet m'a permis de développer un jeu 2D simple en C++ en utilisant la bibliothèque Raylib. Grâce à ce travail, j'ai acquis des compétences pratiques en programmation orientée objet, en structurant mon code avec plusieurs classes (Bike, Obstacle, Game, Menu). J'ai appris à gérer la boucle principale du jeu, les collisions entre objets, le système de timer, ainsi qu'à afficher dynamiquement les éléments à l'écran selon l'état du jeu (menu, jeu en cours, Game Over).

Ce que j'ai appris :

- Organisation d'un projet de jeu avec des classes claires et séparées.
- Utilisation de la bibliothèque Raylib pour créer une interface graphique (affichage de texte, formes, couleurs...).
- Détection de collisions et mise à jour du jeu à 60 FPS.
- Réinitialisation des objets après un Game Over.

Problèmes rencontrés :

- Difficultés lors de l'installation de Raylib et sa configuration avec le compilateur C++ (TDM-GCC et Red Panda Dev-C++).
- Problèmes pour identifier les fonctions spécifiques de Raylib liées à l'interface graphique (`DrawText`, `DrawRectangle`, etc.).
- Quelques confusions initiales sur la gestion du menu et de l'état du jeu (jeu vs menu vs Game Over).

VI Cloture

Notre rapport s’appuie sur certaines ressources bibliographiques , que nous citons :

- https://fr.wikipedia.org/wiki/Raylib#:~:text=Raylib%20est%20une%20biblioth%C3%A8que%20logicielle,orient%C3%A9e%20vers%20le%20d%C3%A9veloppement%20d'

- https://kinsta.com/fr/base-de-connaissances/git-vs-github/

- https://www.ibm.com/docs/fr/i/7.5.0?topic=functions-main-function