TP01A – LoopEngine

# Intro

Vous allez développer la librairie LoopEngine et peu après, vous allez créer un petit jeu avec !

# Règles

## Équipes

2 personnes.

## Remise

Votre dossier git en entier, **incluant les fichiers Raylib**, sur Léa.

# Étape 1 – Résumé de la tâche

Votre tâche consiste à faire l’implémentation de certaines classes de LoopEngine dont les déclarations vous sont fournies.

Aucun membre public des classes fournies ne doit être modifié.

Vous pouvez ajouter des membres comme vous en avez besoin (incluant des publiques).

Les spécifications de chaque méthode et classes sont indiquée dans les fichiers fournis.

## Pourquoi ?

Nous ferons un jeu en utilisant LoopEngine immédiatement après la remise.

# Étape 2 – mise en place

## Visual Studio et raylib

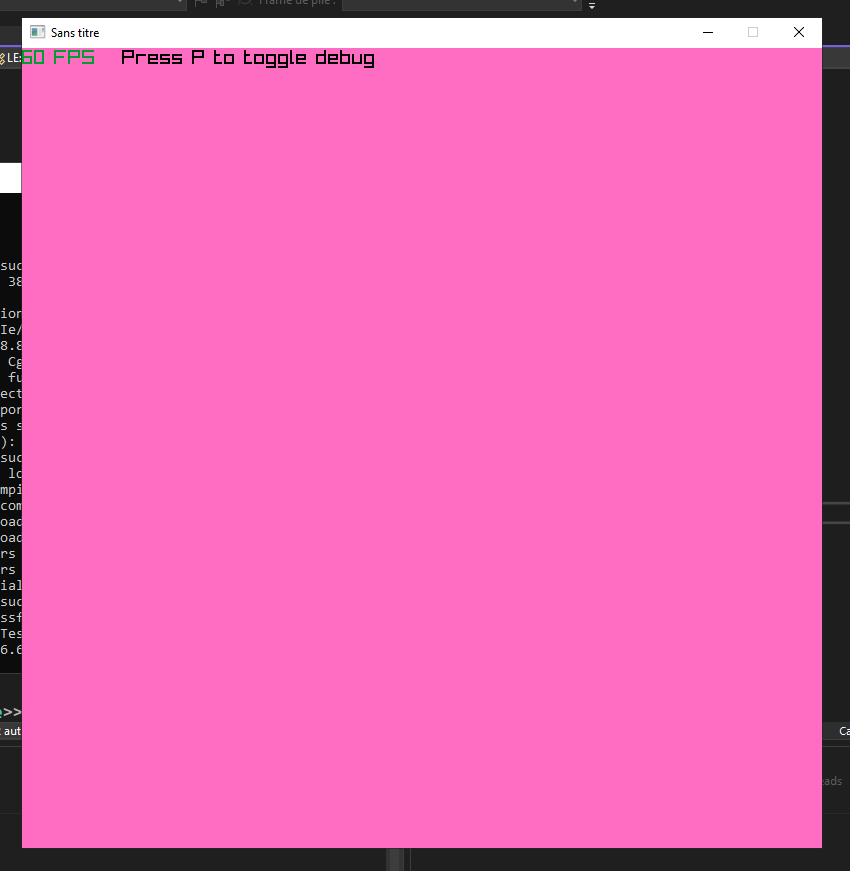
1. Créer un nouveau repo Git et un gitignore approprié. Ajoutez vos collaborateurs.
2. Ajoutez raylib comme sous-module Git.
3. Créer une solution Visual Studio.
4. Créez le projet TP01A à partir de votre patron Raylib
   1. Assurez-vous que tout fonctionne avant d’aller plus loin !

## Fichiers fournis

1. Ajoutez les fichiers que je vous fournis dans votre projet.
2. Le fichier main.cpp doit être à la racine et les autres, dans le dossier LoopEngine
3. Si vous tenter de compiler, vous aurez des erreurs de l’éditeur lien : il manque des définitions de méthodes. Créez toutes les définitions manquantes !

# Étape 3 – Débuter le développement

Le main.cpp fourni est un **exemple** minimaliste de comment utiliser LoopEngine via la fonction TestLoopEngine qui affichera ceci **quand votre code sera complété**.



Pendant le développement, vous bonifierez sans cesse le fichier main.cpp avec vos nouvelles fonctions test. Un exemple est fourni.

# Étape 4 – Classes à compléter

L’idée est de résoudre les problèmes mais en respectant au minimum l’interface publique fourni. C’est comme ça qu’on travaille en entreprise !

## Logger.h

Le logger ressemble à celui fait à l’exercice. Il doit supporter l’affichage des messages à l’écran de jeu. Vous afficherez en permanence les 3 derniers messages dans le bas à gauche lorsque ce mode est activé. Une boîte grise semi-transparente sera affichée en bas et l’écriture des messages sera noir semi-transparent.

## GameObject.h

GameObject2D a deux responsabilités : gérer une Position et les informations pour une boîte de collision.

## Scene.h

Est une fenêtre de jeu (menu, jeu, écran de pointage, écran de partie terminée), etc.)

Contient une collection de GameObject2D ainsi qu’un nom. Pour la plupart des méthodes, va simplement appeler les méthodes du même nom sur GameObject2D. Il faut pouvoir ajouter/retirer des GameObject2D également.

## Engine.h

Contient un dictionnaire de chaînes de caractères vers des Scene.

Quand la méthode StartGame est appelée, on ouvre une fenêtre Raylib et on débute la boucle while dont vous êtes familier. Le string passé permettra d’activer la première scène.

### Scène active

Vous devez avoir une variable contenant la scène active. La scène active celle qui seront appelées dans la boucle while (voir les méthodes indiquées dans le Scene.h).

### Changer de scène active

Lorsqu’on change de scène active…

* + - Si une scène est déjà active, on appelle Unload() dessus.
    - On appelle Load() sur la scène qui vient d’activer
    - On change la scène active pour la nouvelle.

# Étape 5 – Sujet avancés

## Service locator

## Collision

## Commandes