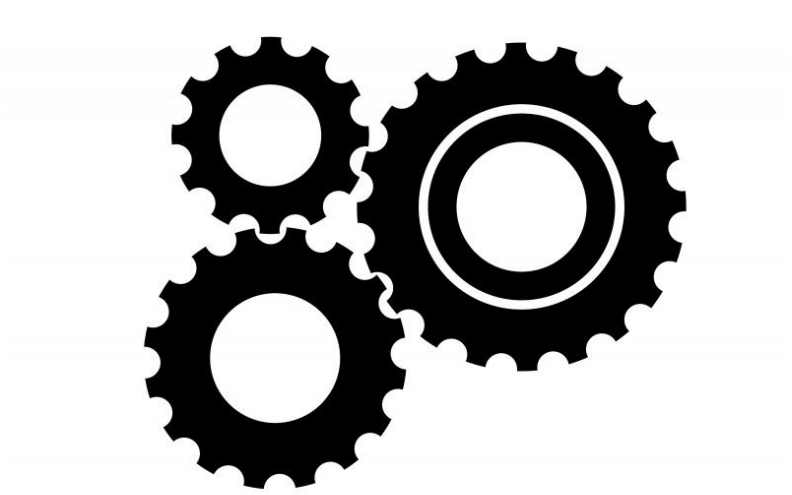
Conception d'un moteur de jeux vidéo

Rapport de Projet



Document réalisé dans le cadre du cours de « Principes des moteurs jeux »

Thomas	Stievenard	STIT31079507
Pierre-Alexandre	N'Guessan	NGUP14129609
Rénald	Morice	MORR14029503
Théo	Debay	DEBT17049500
François	Edorh	EDOF19059507

UQAC 2017/2018

Sommaire

1. Architecture du moteur **3**
1.1. Type d’architecture 3
1.2. Système de logique 3
1.3. Système de rendu 3
1.4. Système de physique 3
1.5. Système audio 3
1.6. Gestion des inputs 4
1.7. Fonctionnalité de sauvegarde/chargement des scènes (si c’est fini avant de rendre) 4
1.8. Système réseau (si c’est fini avant de rendre) 4

2. Caractéristiques **4**

3. Post-Mortem **4**
3.1. Ce qu’il c’est bien passé 4
3.2. Ce qu’il c’est moins bien passé 5
3.3. Améliorations possibles 5

1. Architecture du moteur

1.1. Type d'architecture

Le moteur créé par notre équipe repose sur l'architecture bien connue des « Entités - Composants - Systèmes ». À titre de comparaison, cette architecture est celle que l'on retrouve lorsqu'on utilise Unity 3D. Le moteur a été créé en utilisant C# comme langage de programmation.

Les sections suivantes vont introduire les systèmes implémentés et autres fonctionnalités.

1.2. Système de logique

Un système de logique « LogicSystem » a été implémenté au moteur. Chaque composant souhaitant incorporer de la logique devra implémenter l'interface « ILogicComponent ».

1.3. Système de rendu

Un système de rendu « RenderSystem » a été implémenté au moteur. Chaque composant souhaitant pouvoir réaliser du rendu devra implémenter l'interface « IRenderComponent ».

Ce système s'appuie sur la librairie « OpenGL » grâce au wrapper C# « OpenTK » (https://github.com/opentk/opentk).

1.4. Système de physique

Un système de physique « PhysicSystem » a été implémenté au moteur. Chaque composant souhaitant pouvoir intégrer de la physique devra implémenter l'interface « IPhysicComponent ».

Ce système s'appuie sur la librairie « JitterPhysics » (https://github.com/mattleibow/jitterphysics).

1.5. Système audio

Un système audio « AudioMaster » a été implémenté au moteur. Chaque GameObject souhaitant pouvoir émettre du son devra inclure le composant « SpeakerComponent ».

Ce système s'appuie sur la librairie « FMOD » (https://www.fmod.com/).

1.6. Gestion des inputs

L'ensemble des inputs utilisateur sont gérés via la classe « Input ».

La gestion des inputs s'appuie sur le wrapper C# « OpenTK » déjà introduit précédemment.

1.7. Fonctionnalité de sauvegarde/chargement des scènes

La sauvegarde des scènes est directement gérée dans la classe « Scene » avec la génération d'un fichier JSON dans le dossier « Scenes ». Cette sérialisation est rendu possible grâce à la technologie « Json.Net ».
Le chargement d'une scène se fait à partir du « SceneManager ».

2. Caractéristiques

Voici les caractéristiques de notre moteur de jeu :

- Langage utilisé : C#
- Multiplateforme (Windows, Linux, macOS)
- Modulable grâce à la structure « entités - composants - systèmes » en facilitant l'ajout de nouveaux systèmes par exemple

3. Post-Mortem

3.1. Ce qu'il c'est bien passé

- La mise en place de l'environnement de travail avec l'utilisation de Git comme logiciel de gestion de versions décentralisé
- Présence des systèmes et fonctionnalités de base que nous souhaitions intégrer à notre moteur de jeu

3.2. Ce qu'il c'est moins bien passé

- Communication satisfaisante mais pas toujours optimale
- Parfois difficile à trouver de l'aide concernant certaines librairies C# qu'on a utilisé (communauté un peu moins active que pour le c++ par exemple)
- Difficile de stabiliser l'avancement du projet dans le milieu du trimestre dû au changement involontaire de professeur
- La librairie "Network" n'était pas un bon choix pour essayer d'implémenter une partie réseau. Les tutoriaux sur le site du framework ne semblaient pas fonctionner comme prévu.

3.3. Améliorations possibles

- Optimiser la communication
- Comparer l'activité des communautés d'une technologie avant de la choisir
- Exploiter plus en profondeur les librairies utilisées afin d'offrir des composants plus complets
- Effectuer les différents « TODOs » et FIXMEs » disséminés à travers le code