INTRODUCTION

La partie théorique de l'examen s'appuie quasiment entièrement sur les chapitres à lire dans le livre « *Game Programming Patterns* » de Robert Nystrom.

Certains sujets ne seront **pas directement abordés** dans le cours et c'est à vous de lire les chapitres et de poser des questions au besoin.

Lecture :

Robert Nystrom, Game Programming Patterns, Architecture
https://gameprogrammingpatterns.com/architecture.html

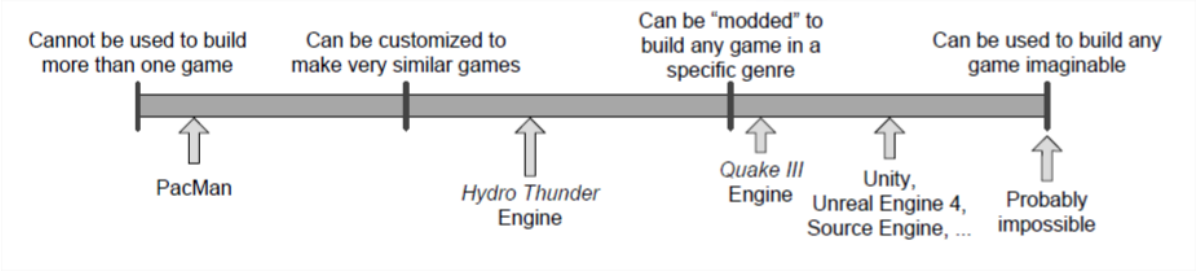
Engin de jeu

Le terme « *Game Engine* » est apparu dans le milieu des années 90 en faisant surtout référence aux jeux « *first-person shooters* » de id-Software : Doom et Quake.

Ces jeux étaient si populaires que plusieurs studios achetaient à id-Software une licence leur permettant d'utiliser le code de base dans leur propre jeu, leur permettant d'accélérer le développement et d'utiliser de la 3D (*qui était révolutionnaire à l'époque*). La séparation claire entre les sous-systèmes génériques et le jeu était nécessaire pour permettre cette flexibilité et donner une valeur ajoutée à leur code. C'est avec cette attitude que Quake III Arena et Unreal furent développé et c'est le défi que nous nous lançons dans ce cours.

🔗 Id-Software ont rendu plusieurs jeux et outils disponible https://github.com/id-Software

La ligne entre le code de jeu et celui de l'engin est souvent floue. Règle générale, la logique de jeu spécifique à un cas, les règles « *hard codé* » et les cas spéciaux rendent le code difficile, voire impossible à réutiliser.



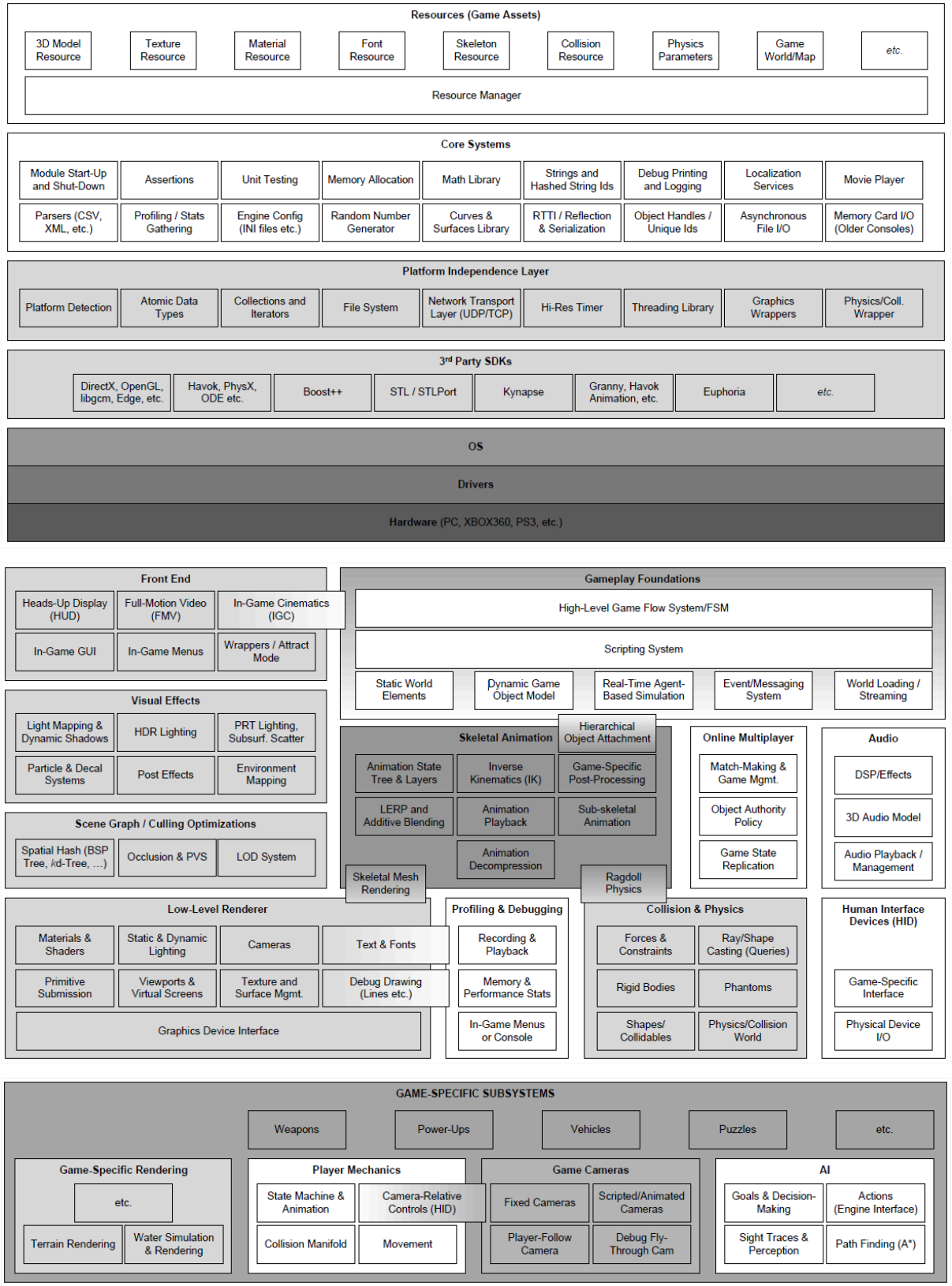
Plus un engin est générique, moins il est optimal pour exécuter un type précis de jeu.

En général, un engin est composé d'un ensemble d'outils (*éditeurs*) et de codes. L'architecture de la partie de code d'un engin est typiquement séparée en sous-systèmes que l'on appelle des gestionnaires (*ou services*) qui ont chacun leurs responsabilités d'une partie du jeu. Le tableau suivant énumère des exemples de système générique qui pourrait se retrouver dans n'importe quel jeu.

Services	**Responsabilités**
Journalisation	Aide au débogage en signalant tout problème aux programmeurs
Graphiques	Affichage des images du jeu
Input	Gestion des contrôles
Audio	Jouer des sons et de la musique
Scene	Organisation des objets dans une partie du jeu
Resources	Chargement des ressources du jeu
Reseau	Inter-connectivité entre joueurs
Mémoire	Gestion de la mémoire et prévention de fuites
Physique	Simulation de la physique dans le jeu

Tous ces services ne dépendent pas d'un jeu précis et pourraient, si le code est bien fait, avec une attention à la réutilisabilité, être partagés dans plusieurs projets. Que ce soit dans « *World of Warcraft* » ou dans « *Chess Master* », les deux jeux doivent afficher des images, jouer des sons et gérer des ressources.

Un engin moderne implique beaucoup de systèmes et est hors de portée de ce cours. L'image suivante vous montre les systèmes les plus communs d'un engin :



Je conseille de lire tout le livre. Je suis conscient que vous n'aurez pas le temps durant la session de tout le lire, alors voici ce que je vous propose. En premier lieu, nous allons mettre la priorité sur les chapitres qui servent au cours. Les autres chapitres sont tout aussi importants, mais vous pouvez les lire plus tard (ou c'est de la matière que vous avez déjà vu dans l'AEC). Les chapitres, que j'ai annoté « skip », sont pour plus tard (ou au besoin pour consultation).

Architecture, Performance, and Games	**(à lire dans ce cours) (1)**
Command	(fondamental)
Flyweight	(très utilisé, AEC)
Observer	**(à lire dans ce cours)**
Prototype	**(à lire dans ce cours)**
Singleton	**(à lire dans ce cours) (3)**
State	**(à lire dans ce cours)**
Double Buffer	(skip)
Game Loop	**(à lire dans ce cours) (2)**
Update Method	**(à lire dans ce cours) (4)**
Bytecode	(skip, AEC)
Subclass Sandbox	**(à lire dans ce cours) (6)**
Type Object	(skip)
Component	**(à lire dans ce cours) (5)**
Event Queue	(skip)
Service Locator	**(à lire dans ce cours) (3)**
Data Locality	(skip)
Dirty Flag	(skip)
Object Pool	(fondamental, AEC)
Spatial Partition	(très utilisé)

Lecture des chapitres
Quiz (pas d'examen)
Les quiz peuvent contenir des questions sur les notes de cours ou sur le code de l'engin, mais sont principalement en lien avec la théorie contenue dans le livre.

Choix de jeux
Remise du jeu
Note de cours
Suivre dans le cours
Programmer sont engin (risques)
Backup (sources)
Point bonus (dans ton engin)

Challenge (vidéos)

- Durant les cours, je vais coder avec vous. À certains moments, je vais vous laisser coder seul, question de vous pratiquer. J'appelle ces moments des «challenge». Si vous ne le réussissez pas (ou ne le faites pas), une solution vidéo vous sera remise à la fin du cours.

Notez que c'est possible que le code d'un vidéo differ de ce que j'ai fait en classe ou dans le code du cours (notes, sources, etc). Mais, ca reste une possible solution au challenge.

Laboratoire en différentes étapes.

- Avant de commencer un laboratoire, vous aurez accès à un build qui démontre le résultat à atteindre. Il ne doit pas avoir d'ambiguïté sur ce qu'il doit être fait lors des laboratoires.

- Le code de l'engin vous est remis après chaque laboratoire. Le but étant de s'assurer que si vous ne réussissez pas bien la partie engin du cours, c'est encore possible de réussir la partie jeu.

- Il est bien sûr idéal de tenter de faire votre engin vous-même, sans avoir recours au code de l'enseignant. Des points bonus sont attribués pour vous encourager de prendre de l'expérience (dans la remise finale).

Exemple de projet réalisé par d'ancient étudiants

Erreur commune dans ce cours :

- Ne pas mettre la priorité sur une boucle;
- Chercher des ressources graphique avant de commencer;