# GÉNIE LOGICIEL – REALISATION D'UN JEU VIDEO

# QU'EST CE QUE LE JEU VIDEO ?

- Un art : 10e art en France depuis 1993
- Une industrie croissante : 49 M$ en 2008 => 100 M$ en 2017
- Un support pour véhiculer un message
- Buts :
  - **Ludique : Divertir, Amuser**
  - Sérieux : Entraîner, Comprendre
  - Educatif : Apprendre

# QU'EST CE QU'UN JEU VIDEO ?

- Un **jeu vidéo** est une application informatique **multimedia** doté d'une interface utilisateur (au sens large) permettant une **interaction** humaine ludique et amusante entre un **joueur** et un **environnement virtuel**.
- 2 notions capitales:
  - Interactions : Le joueur agit sur l'environnement (entrée) / L'environnement agit sur le joueur (sortie)
  - Médias : Le véhicule de l'information / l'interaction

# QU'EST CE QUE LE JEU VIDEO ?

Intérêts du point de vue scientifique et informatique:

- Traitements complexes de multiples informations au sein d'une même application
- Intersection de différents domaines de l'informatique(graphique, physique, sonore, intelligence artificielle, réseau…)
- Intersection entre différents domaines et des dizaines de corps de métiers différents

- Depuis quelques temps :
  - Utilisation de plusieurs supports (pas forcément electronique) => **Transmedia**
  - Etude de l'impact de nouvelles réalités : virtuelle, alternée, augmentée, mixte….

# QU’EST CE QU’UN JEU VIDEO ?

- Pour faire un jeu vidéo, il faut donc un support : images, textes, sons…

- Une interaction : Qu’est ce que l’utilisateur doit faire ? Comment les actions modifient l’environnement virtuel ?

C’est ce qu’on appelle le **Gameplay**.

# DU POINT DE VUE DU GENIE LOGICIEL

- Le Gameplay représente donc la couche métier, ce que fait le programme informatique
  - L'utilisateur final : Le joueur…
  - La finalité : Le scénario, le but du jeu…
  - Les fonctionnalités : Les actions pour réaliser le but, le comportement des éléments de l'univers virtuel, la transformation des médias…

=> Les besoins sont regroupés dans le Game Design Document

# QU’EST CE QU’UN JEU VIDEO ?

# EXEMPLE

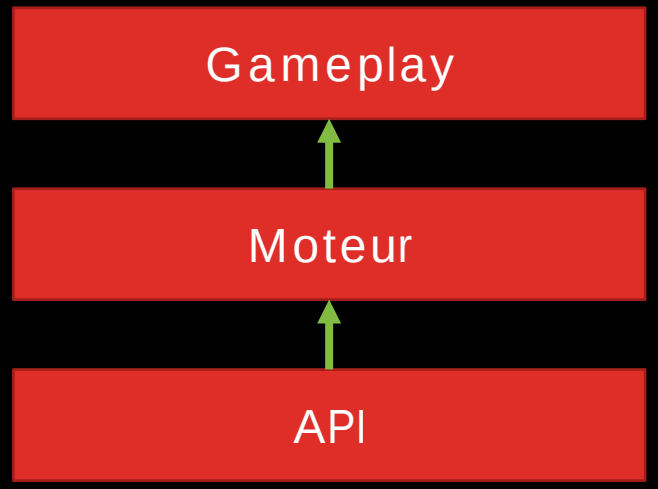
- Assassin’s Creed (Ubisoft)
  - Objectif : Raconter une histoire ludique dans le contexte des Croisades
  - Les médias : Animations, sons, décors…
  - Les fonctionnalités :
    - Du point de vue intérieur : Déplacer le héros, combattre, sauter, fuir…
    - Du point de vue extérieur : Déplacer la caméra, lire des séquences animées, sauvegarder une partie….

# METHODOLOGIE DE DEVELOPPEMENT

- Différenciation entre le Gameplay (ce qui se passe dans l'univers) et le Traitement (les opérations informatiques sur les médias)
- Le Gameplay est **spécifique** au jeu
- Le Traitement algorithmique est générique => Moteur de jeu
- Le manière d'utiliser le support => API

# METHODOLOGIE DE DEVELOPPEMENT

- Architecture en 3 couches
  - L'API : Interaction directe avec le matériel (OpenGL, DirectX...)
  - Moteur : Ensemble de fonctionnalités génériques assemblées en module (Algorithme d'affichage....)
  - Gameplay : Le jeu en lui-même et ses règles



# LA COUCHE MOTEUR

- Développement du moteur => Discipline à part
- Consiste à écrire les algorithmes pour traiter et transformer l'information ou un média
- Plusieurs modules
  - Graphique : gère l'affichage de l'environnement virtuel et sa représentation visuelle
  - Physique : gère les calculs et les interactions physiques
  - Sonore : gère le son
  - Noyau : gère les interactions entre les différents modules

# LA COUCHE MOTEUR

- La couche moteur ne contient **AUCUNE** règle de jeu.
- La couche moteur s'appuie sur l'API qui interagit avec le materiel.
- Objectifs :
  - Réutilisabilité du code
  - Déplacer à un niveau inférieur les routines et les opérations complexes applicables à n'importe quel jeu

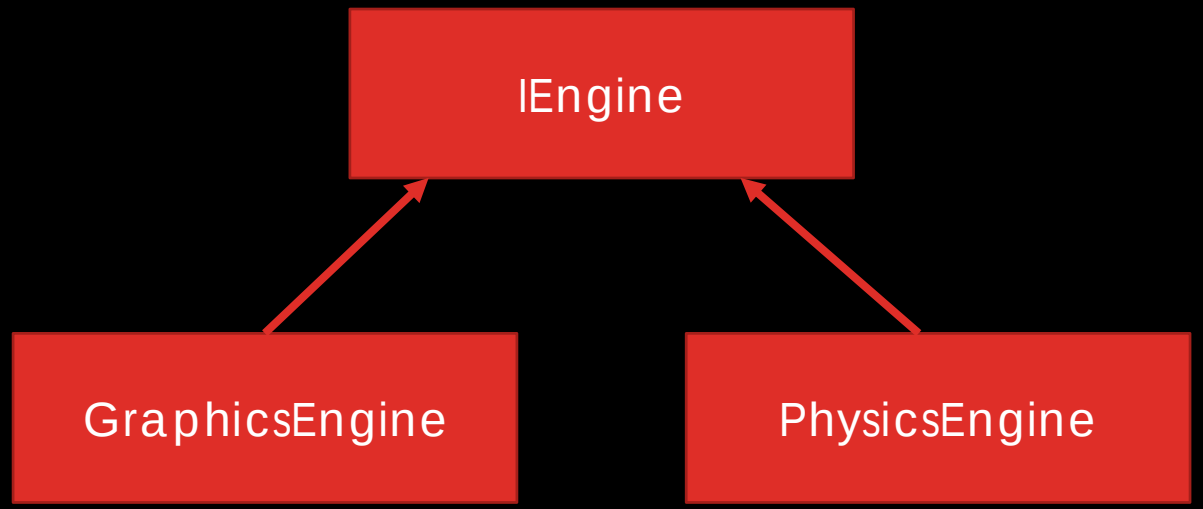
# LA COUCHE MOTEUR

- Spécifier un moteur = Décrire les fonctionnalités et les algorithmes de traitement de l'information

- Exemple : Fonctionnalités du moteur graphique
  - Gérer la représentation de l'espace mathématique
  - Gérer les transformations : Translation, Rotation, …
  - Afficher en s'appuyant sur les fonctions de dessins de base

# CONCEPTION D'UN MOTEUR

- Bien diviser les fonctionnalités en module
- Etablir une architecture modulaire en tirant partie de la POO
- Tirer partie de la programmation **évènementielle** et **multithreadée**



# COUCHE GAMEPLAY

- Ne contient **QUE** les règles de jeu spécifique à un jeu.
- Représentation des éléments du GDD.
- **AUCUNE** routine générique.
- => La couche Moteur peut etre utilisée dans différents projets.
- => La couche Gameplay est **spécifique** à un projet.

# CONCEPTION DU GAMEPLAY

- Identifier les éléments du gameplay pour en tirer leur architecture.
- Exemple:
  - Fonction: Le **joueur** doit **déplacer** un **héros** qui peut être un **monstre carré**, ou un **oiseau**. Il peut aussi **lancer** des **bombes**.

# CONCEPTION DU GAMEPLAY

- Joueur : Représentation informatique de l'utilisateur dans l'univers réel qui manipule une manette.
- On crée une classe **Joueur** qui contiendra donc les éléments de profils, la gestion des commandes...
- Cette classe s'appuiera sur le Moteur d'entrée sortie.

# CONCEPTION DU GAMEPLAY

- Un héros => Monstre carré ou un oiseau
- 1 interface générique / classe abstraite : Le héros qui contiendra deux opérations : bouger, lancer des bombes
- 2 classes filles : Monstre carré, Oiseau
- S’appuie sur la classe Entity du Moteur Graphique, PhysicEntity du Moteur Physique

# CONCEPTION DU GAMEPLAY

- Bombe : Différentes représentations possibles.
- Best practice du génie logiciel : Utiliser les Design Pattern et les « classes-bouchons »
- 1 classe abstraite / 1 Interface : Ibomb (Au cas où le GDD est amené à évoluer)
- 1 classe réelle : CBomb qui représente la bombe définit dans le GDD

# DÉVELOPPEMENT D'UN JEU

- Processus itératif en parallèle : Le moteur et le gameplay évolue par itérations en même temps, mais de manière indépendante

- On introduit au fur et à mesure les fonctionnalités dont on a besoin

# EXEMPLE

- Fonction: **Afficher** un **héros** qui **avance**.
- Besoin Moteur :
  - Algorithme d’affichage, gestion mathématique de l’espace
- Besoin GP :
  - Faire avancer un héros

# EXEMPLE

- Spécifications moteur:
  - Définir un algorithme pour afficher une image (généralement fourni par l'API)
  - Définir une représentation spatiale : une Entitée = un Point + Un vecteur
  - Définir les transformations : Translation, Rotation…

  - Note : Le Moteur peut parfois se résumer à une simple réencapsulation !

# EXEMPLE

- Spécifications GP:
  - Un héros qui avance : Définir les attributs du héros, définir sa façon de se déplacer (courbe linéaire, logarithmique…)
- Conception Moteur: définir l'architecture pour le moteur graphique (et éventuellement ce dont on aura besoin pour le réaliser)
- Conception GP : définir l'architecture pour traduire les spécifications GP (une classe héros qui héritera de la classe Entity du Moteur par exemple)

# EXEMPLE

- Tests du moteur : On teste les fonctionnalités **du moteur** => Est-ce que le moteur affiche correctement une image si je le lui demande ? Est-ce qu'il le déplace correctement ?

- Tests du gameplay : On vérifie et on valide le comportement **du jeu** par rapport aux règles spécifiées : Est-ce que le héros se déplace de la bonne façon ?

# EXEMPLE

- On réitère jusqu'à arriver au jeu complet.
- Dans la vraie vie : moteur et gameplay évolue en parallèle => Deux cycles de vie différents
- Dans votre projet : un livrable = une évolution moteur et/ou une évolution gameplay