# Projet GL : Jeu vidéo (1)

L'objectif de ce projet est de réaliser un logiciel de qualité en utilisant les méthodes et techniques issues du génie logiciel. L'une des spécificités du domaine du jeu vidéo concerne les changements continuels de besoins et de spécifications dans des délais extrêmement courts. Ces changements rapides et soudains de besoins et de spécifications rendent très délicate l'application des méthodes de développement traditionnelle. Les méthodes itératives, telles que SCRUM ou XP, bien que semblant convenir à ce type de logiciel, peinent à être efficaces si elles ne sont pas appliquées avec une grande rigueur et une grande maîtrise. Même si les exigences sont moindres que celles requises par le développement de systèmes critiques, réaliser un jeu vidéo dans un délai court (7 semaines) est donc un bon entraînement pour la mise en pratique des méthodes itératives, tout en couvrant un large spectre de domaines et de connaissances en informatique (programmation graphique, programmation sonore, programmation physique, réseau, IA (Intelligence Artificielle) programmation parallèle, etc.).

## 1. Contexte

L'objectif consiste donc à réaliser en 7 semaines un prototype de jeu vidéo que les étudiants pourront comparer avec les produits existants sur le marché, leur donnant ainsi une première idée du travail et des contraintes nécessaires pour réaliser un produit commercial de ce type.

## 2. Architecture traditionnelle

Les jeux vidéo s'appuient en grande majorité, traditionnellement, sur une architecture à 3 couches :

- les bibliothèques bas niveau (développées par les constructeurs) fournissent une interface entre le matériel et les composants moteurs ;
- le moteur de jeu (*engine*), divisé en plusieurs modules moteurs, fournit les routines génériques élémentaires (core, gestion de l'affichage et des effets graphiques, gestion des sons, gestion du réseau, etc.) ; le moteur de jeu se veut en général générique et réutilisable ;
- le *gameplay* implémente les règles et mécaniques de jeu définit dans le *Game Design Document (GDD)* – un document très proche d'un dossier de spécifications.

La démocratisation du développement de jeu vidéo au niveau amateur et indépendant a mené à la création de bibliothèques de fonctions dans divers langages qui reprennent plus ou moins les deux couches les plus basses, ramenant ainsi le développement du jeu vidéo plus ou moins à l'implémentation de la troisième couche. Il est toutefois intéressant de se confronter à la réalité et de leur faire travailler en partie sur la couche *engine* (en s'appuyant sur des librairies multimédia existantes telles que Qt ou LWJGL (middleware) qui fournissent des routines médianes entre ces couches), et également sur la couche *gameplay*.

## 3. Enoncé du besoin

Fort du succès d'un précédent jeu (http://store.steampowered.com/app/280830/), le client est un éditeur de jeu vidéo reconnu, représenté par M. HAMRI et M. YACOUB, qui demande aux étudiants

d'AMU de réaliser, dans des délais extrêmement courts, un jeu vidéo d'arcade footballistique en deux dimensions. Le rendu graphique se veut très simpliste. Le fait qu'il s'agisse d'un jeu d'arcade permet de s'affranchir de règles réelles, diminuant la complexité du produit. Notamment, le jeu consiste en une série de match dans lequel s'affronte deux équipes de 3 joueurs (un gardien de but, un défenseur et un attaquant) avec des règles de jeu simplifiés :

1. Il n'y a pas de fautes considérées.
2. Si le ballon sort hors des zones du terrain en 6m ou corner, il revient immédiatement dans les pieds du gardien adverse.
3. Il n'y a pas de touches (les murs font rebondir la balle).
4. Il n'y a pas de gestion de fatigue (mais peut-être une jauge permettant de sprinter).
5. Le gardien de but ne peut pas être gérer par le joueur humain.

Un premier mode de jeu consistera à affronter un joueur humain contre l'ordinateur.

Un mode de jeu à deux joueurs (en réseau ou via deux *gamepads*) est possible.

Si d'autres modes de jeu peuvent être imaginés dans le futur, l'éditeur souhaite en priorité que ces deux modes de jeu soient implémentés.

Les équipes sont personnalisables (choix des joueurs parmi une liste de joueurs disponible avec des caractéristiques différentes, choix des couleurs du maillot).

## 4. Moteurs

Comme dit précédemment, le moteur est subdivisé en sous-modules moteurs fournissant chacun des fonctionnalités génériques utilisables par la couche *gameplay*. La majeure partie de ces modules seront simplement une ré-encapsulation des méthodes fournies par les middlewares utilisés. Il peut être intéressant de découvrir et découper le moteur, tout en étant contraignant dans les techniques employées. Par exemple, le moteur d'IA peut fournir tout un tas de fonctionnalités basés sur les moteurs d'inférence, les réseaux de neurones, etc. Afin de pouvoir réaliser le projet sur la durée imparti, il est donc nécessaire de limiter le champ des algorithmes pouvant être implémentés. A titre indicatif, voilà une liste des modules moteurs traditionnels :

1. Le *core-kernel* : fournit les routines et les structures standards utilisées par les autres modules. Le *core-kernel* s'occupe également de la synchronisation des modules moteurs (séquentiels ou parallèles).
2. Le moteur graphique : fournit les routines d'affichage, d'effets spéciaux, d'animation et de gestion de la scène. Habituellement, une scène de jeu vidéo est partitionnée dans une structure spatiale permettant d'optimiser le rendu et de simplifier un certains nombres de calculs.
3. Le moteur physique : fournit les routines de gestion physique (collision, calcul de vitesse, etc.).
4. Le moteur IA : fournit les routines de gestions d'agents intelligents (*pathfinding*, moteur d'inférence logique, etc.).
5. Le moteur sonore : fournit les routines de gestion et de synchronisation du son.

6. Le moteur UI (*User Inteface*) : fournit les routines permettant de gérer les éléments d'interfaces (menus, widgets,etc.) et l'interface utilisateur (HUD *Head-Up Display* qui renseigne le joueur sur son environnement).
7. Le moteur réseau : fournit les routines permettant de gérer les interactions réseaux.

## 5. Assets

Le commanditaire du projet s'engage à fournir aux équipes de production ce dont ils ont besoins pour développer le projet en termes d'*assets* graphiques et sonores, si besoin. De nombreux *assets* libres de droit peuvent également être trouvés sur le net.

## 6. Références

Football Manager : http://www.footballmanager.com/manual/?game_id=38&id=604&locale=fr_FR

Metegol : http://store.steampowered.com/app/280830/

L'histoire du foot dans le jeu video : http://www.be-games.be/blog/2011/08/lhistoire-du-football-dans-les-jeux-video/