Projet 2 Génie Logiciel : Jeu vidéo

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 en 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-kernel, 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

Pacman est un jeu vidéo populaire et légendaire des années 80 et continue d'attirer les convoitises des utilisateurs (joueurs), des développeurs et même des chercheurs.
Le client de ce jeu est une entreprise spécialisée dans les jeux vidéo, réprésentée par Ms Hamri et Khemiri. Il demande aux étudiants de l'AMU de développer le jeu vidéo Pacman en deux dimensions.

Le jeu est composé d'un personnage en forme de cercle se déplaçant à l'intérieur d'un labyrinthe rempli de points de fruits qui permettent de scorer une fois que le personnage les a avalés. Quatre personnages ennemis dits fantômes sillonnent et le moindre contact conduit le joueur à l'échec. Sauf s'il est doté de super pouvoir, il pourra les abattre. Malheureusement ce super pouvoir est momentané et le Pacman doit par la suite les éviter. Le joueur passe d'un niveau à un autre une fois tous les points et les fruits consommés.

Ce jeu peut être chronométré à l'aide d'un sablier. Par conséquent, le joueur est déclaré automatiquement perdant à la fin du temps alloué si la partie n'a pas été réussie. Le nombre de fantômes présents dans le labyrinthe est paramétrable et peut être changé selon le niveau de difficulté choisi.

Malgré que le jeu Pacman est bien connu pour être joué en mode solo, nous pouvons imaginer un mode duo où un joueur contrôle le Pacman et un deuxième joueur manipule le ou les fantômes présents. Ce mode de jeu particulier peut s'avérer utile pour la validation du jeu à développer,

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 et les librairies 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 (qu'ils soient 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 (graphe de scène, arbres…) 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.

6. 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.