**Travail pratique 1**

**Laborat – Rapport de projet**

**Analyse textuelle**

Laborat est un projet *pygame* ou le Laborat (personnage joueur), doit s’échapper, trouver le fromage (condition de victoire) et éviter des zombies (condition de défaite). Les trois types d’acteur actif dans le jeu seront représentés par des carrées de différente couleur. Laborat est le seul acteur pouvant bouger. Tous les types d’acteurs doivent pouvoir enregistrer des collisions avec Laborat. Les collisions entre Laborat et le fromage ou les zombies terminent le jeu. L’arrière-plan doit changer de couleur dépendamment de l’état du jeu.

**Conceptualisation**

Notre version de Laborat va implémenter une architecture de type *MVC*. *MVC* nous semblait être approprié dans pour le type de jeux a construire comme on peut facilement trancher les élément modèle des élément de contrôle usager et visuel.

Le programme comprendra 3 modèles principaux : *game*, *actor* et *actor\_factory*.

*Game* est le modèle central du jeu. C’est dans cette classe que l’on retrouve l’initialisation des éléments centraux du jeu à son bon fonctionnement (*window*, *listeners*, *actors*, clock, ect.). Comme cet objet central s’avère absolument essentiel à la bonne coordination des opérations du logiciel et qu’un seul et nécessaire, un patron de type singleton a été choisi pour sa conceptualisation. C’est aussi la classe *game* qui contient la boucle de jeux principale (*run*), applique les *input* capturer par le *game\_listener* (*action\_interpreter*) et gère les conditions de victoire.

Actor représente tous les éléments dynamiques du jeu. Cette classe contient les informations sur la couleur, position et état de l’acteur et ses méthodes de type *get/set*. De *actor* découle 3 sous-classes : *Player* pour le personnage joueur, *Fromage* pour l’objectif et *Zombie* pour les obstacles. Pour le moment, la seule différence entre ces trois classes est leurs couleurs, mais différente méthode et variable pourront y être ajoutées si besoin est.

*actor\_factory* est la classe en charge de l’initialisation des objets de type *actor*. Fortement inspirée au niveau de sa position dans l’architecture du patron de fabrique, notre classe *actor\_factory* s’en éloigne toute de même. Comme les positions des acteurs et leur quantité sont fixées, notre fabrique n’instancie pas dynamiquement les sous-classes de *actor* mais bien de façon procédurale. Considérant de futures versions de Laborat à l’horizon, cette classe a été intégrée à l’architecture initiale par souci d’évolutivité.

Laborat ne comporte qu’un seul contrôleur, *game\_listener*. Cette classe est en charge de pousser les événements de clavier vers le modèle *game*

Finalement, le programme aura deux éléments vue : *Window* et *IconesActeurs*. *Window* prendra les informations du modèle **game** et les rendra à l’écran, elle est aussi en charge de l’affichage et de la boucle de mise à jour des icônes. Sa méthode *end\_game* est responsable de refléter les victoires et défaites. *IconesActeur* quant à elle hérite de la classe *pygame sprite*. Elle contient l’objet *Pygame* *rect* qui défini nos acteurs visuellement et défini leur collision.

**Exécution**

Grâce à l’initiative et enthousiasme de notre collègue Clément, nous avons lancé notre première réunion de conceptualisation avec une longueur d’avance. En effet, Clément avait déjà jeté un premier jet (fichier *Laborat.py* dans les plus vieux commit). De ce prototype du jeu, nous avons commencé par diviser l’architecture vers un modèle *MVC*.

Notre première réunion avant sprint nous a permis d’éviter de frapper des blocages pour la majorité du projet. C’est à cette première rencontre que le choix des patrons approprié et le travail du premier sprint divisé. Tom et Olivier ont travaillé sur les modèles, Tym et Clément sur les éléments de vue et de contrôleur.

Étant tous des adeptes du *pair programming*, la division en équipe de deux nous semblait la meilleure. Les deux équipes ont été confrontées à des défis différents. L’équipe en charge des modèles s’est vu travailler davantage sur l’implémentation du singleton et s’assurer que la boucle de jeux reste fluide à l’exécution. L’équipe en charge de la vue s’est vue confronté a un apprentissage plus poussé des divers méthodes et classe de *pygame* de façon a bien implémenté les mouvements et les méthodes de sprite.

Notre deuxième sprint a principalement consisté à l’intégration de ces diverses composantes au projet. Ce travail s'est fait pour la majeure partie avec le groupe complet et s’est déroulé quasiment sans anicroche grâce à notre planification.