**Travail pratique 3**

**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 quatres types d’acteur dans le jeu sont représentés par des animation qui sont composé d’un succession d’image qui est propre à chaque acteur. Laborat et les zombies sont les seuls acteurs pouvant bouger. Il y a également des murs que ni Laborat ni les zombies ne peuvent traverser. 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. La fenêtre doit s’immobiliser et afficher un message dépendamment que le joueur perde ou qu’il gagne. La fenêtre est une grille de 9 lignes et 16 colonnes, chaque acteur occupe une case. Les acteurs immobiles et mobile sont séparé en 2 catégories pour l’affichage et ceux-ci sont affichés sur deux différentes couches pour amélioré la performance du jeu.

**Conceptualisation**

Notre version de Laborat implémente une architecture de type *MVC*. *MVC* nous semblait être approprié dans pour le type de jeux à construire comme on peut facilement trancher les éléments venant du model des éléments de contrôle usager et visuels.

Le programme comprend 8 modèles principaux : ***game***, ***actor***, ***actor\_factory****,* ***etat****,* ***tile****,* ***command******pathfinder***et ***coord****.*

*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***, ***actor****s*, **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 jeu principale (*run*), applique les *input* capturer par le ***game\_listener*** et gère les conditions de victoire. Théoriquement, c’est ici qu’aurai dû aller l’attribut *couche* qui permet une actualisation des élément du jeu par ordre de priorité*. C*ompte tenu de l’architecture de notre code, il n’aurai pas eu beaucoup de sens que les couches d’affichage appartiennent à Jeu comme les changements à la grille de jeu sont enregistrés dans *etat*.

**Actor** représente tous les éléments du jeu. Cette classe contient les informations relatives à l’affichage. **Actor** a en attribut *sprite* qui est l’image le représentant à ce moment ainsi que la *direction* qui sert à savoir dans quel direction l’acteur se déplace ***Actor***possède une méthode abstraite *get\_sprite* qui est utilisé par tous les acteur du jeu pour retourner leur image. **Actor** a aussi une fonction pour extraire les sprites de la sprite sheet. De ***actor*** découle 5 sous-classes : *Player* pour le personnage joueur, *Fromage* pour l’objectif, *Zombie* pour les ennemis , *Wall* pour les obstacles et *Floor* pour le plancher. Chacune de ces sous-classes contiennent au moins une image qu’ils sont responsable d’enregistrer et qui leur donne la possibilité d’affichage. *Player* et *Zombie* ont une méthode *update\_sprite* qui permet de gérer la bonne animation à afficher parmis la liste des sprites disponibles pour les acteurs mobiles. Ces deux classe possèdent aussi une fonction *increase\_counter* qui leur permet de déterminer à quel moment de l’animation qu’elles sont chargées d’afficher. *Wall, Floor* et *Fromage* n’ont pas de méthodes puisque ce sont des acteurs statiques.

***actor****\_f****actor****y* est la classe chargée de l’initialisation des objets dynamiques des objet des sous-classes **actor**s et les retourne avec ses méthodes get.

***état*** est la carte du jeu, c’est là que sont enregistrés des tuiles qui contiennent les acteurs et leurs positions. Le fichier *maps.txt* contient plusieurs cartes prédéfinies, *etat* en sélectionne une au lancement du programme de manière aléatoire. C’est ici aussi que sont enregistrés les élément qui ont été changé durant le tour de jeu pour les passer à ***window***quivient remplacer le rôle des couches. Elle permet aussi de générer la liste des positions invalides pour les acteurs.

***tile*** est une classe représentant les tuiles. Les tuiles servent de contenant pour les acteurs. Quand l’**état** est généré, il contient une grille de tuiles. ***Tile***contient deux types de tuiles : ***Tile****\_ Game* et ***Tile****\_Pathfinding*. **Tile**\_Game sont les tuiles générés et enregistré dans la grille d’état. Elle contient une liste d’Acteur ainsi que ses propres coordonnées. ***Tile****\_Pathfinding* sont des tuiles utilisé par la classe *Pathfinding* pour déterminé les chemins qu’empruntent les rat-zombies.

**Command** contient 3 données : un acteur, une position de départ et une position d’arrivée. Les **command**es sont des objets permettant une interprétation uniforme des événements peu importe leur provenance que ce soit de Player ou Zombie. ***Command***possède aussi une fonction *define\_movement\_direction* qui permet à l’acteur qui émet la **commande** d’actualiser son *sprite* selon la direction ou la **commande** l’emmène.

***Coord*** est une classe utilitaire chargée de la structure coord, un tuple de position (x,y ) ainsi que ses méthodes ***getter/setter.***

***PathFinder*** est la classe responsable de la vérification de mouvement des acteurs mobiles. Elle est aussi responsable de l’IA de rat-zombies.

Laborat ne comporte qu’un seul contrôleur, ***game\_listener***. Cette classe est chargée de pousser les événements de clavier vers le modèle ***game****.* Par contre, les événements de clavier sont retournés à l’état sous forme de commandes qui servent de patron général et qui peuvent être générés par tout acteur mobile.

Le programme un élément : ***Window*** . ***Window*** prendra les informations de l’**état** dans la liste des tuiles changées et les actualiser et les rendra à l’écran, elle est aussi chargée 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.

**Exécution**

L’exécution du travail cette fois s’est passé comme sur des roulettes.

Dans cette étapes du développement, nous n’avons par rencontré d’obstacle majeur. Le développement d’un nouveau pathfinding à entrainé quelques bugs à début de sont implémentations, mais ceux-ci ont vites été règlé. Aussi, les sprites que nous avions implémenté lors de la première étapes, enlevé au cours de la 2e étapes et remis pour cette étapes nous ont occasionné quelques problèmes qui ont été règlé vers la toute fin.

Nous avons d’abord commencé par ajouter les éléments nécessaires au diagramme de conception pour faciliter la division du travail. Ensuite nous nous sommes divisés le travail en fonction de la quantité de programmation estimée: Tom et Olivier ont travaillé ensemble sur le Pathfinding comme ils avaient tout deux implémenté la précédante itération., Tymotheï à travaillé sur l’intégration des spirtes et sur l’Affichage en couche avec son implémentation d’un *glass pane* pour les éléement changeant. Clément s’est chargé du package “**Controller”** et l’implémentation des raccourcis claviers pour quitter le jeu et recommencé la partie ainsi que de la documentation.

En général, cette étape s’est bien déroulé avec une difficulté minimales. Tous le monde ont participé au mieux de leur aptitude et ont été attribué des tâches qui correspondaient à leur forces.