Rapport De Projet

Auteurs :
Guillaume Pierre, Valdrin Salihi

Professeurs Encadrant :
Mr.Skapin, Mr.Peltier

Table des matières

Documentation utilisateur :

Tout d’abord une contextualisation de l’environnement dans lequel ce jeu se trouve.
Nous avons décidé de créer un jeu d’aventure où l’on incarne un chasseur de fantômes qui pourrait s’apparenter très modestement à un jeu vidéo très célèbre du nom de Luigi’s Mansion. Le but est comme nous l’avons dit précédemment, de chasser toute sorte de fantôme au sein d’un château hanté. Il faudra donc pouvoir avancer à travers les salles du château en éliminant les spectres qui vous barreront la route afin de trouver d’où provient cette inquiétante source d’anomalies.

Maintenant, au niveau de l’utilisation du jeu, il se jouera à partir d’un invite de commande donc avec un affichage purement textuel. Le joueur pourra interagir avec le jeu grâce à des commandes qui lui seront constamment proposées afin d’avancer dans ce dernier. Des commandes donc lui seront affiché qui permettront chacune d’avoir des actions définies dans le jeu. Il aura les possibilités standards de se déplacer d’une salle à une autre grâce à la commande « GO location ». Il pourra aussi inspecter ce qui se trouve dans chaque salle avec « LOOK [observables] » ainsi que de prendre ce qui s’y trouve à l’aide de l’action « TAKE takable ». Et donc par extension des deux dernières commandes citées, la commande « USE item1 [item2] » va pouvoir lui permettre de se servir de différents objets existant dans le manoir. Lorsque le joueur sera perdu, l’action « HELP » sera là pour lui rappeler toutes ses actions possibles afin de reprendre sa progression. Il pourra bien évidemment se battre ou à l’inverse se défendre avec l’action « ATTACK ». Lorsque le joueur aura découvert un personnage non-joueur étant le jardinier du château, une nouvelle commande lui sera possible du nom de « GARDENER » lui permettant peu importe sa position dans le donjon de revenir voir le jardinier. Et enfin une des dernières actions et pas des moindres dans l’intérêt du joueur sera la commande « QUIT » qui lui fera sortir de la partie à tout moment.

Documentation développeur :

- Diagramme de classes :

Le diagramme de classes du projet s'articule principalement autour de cinq grandes classes, ces dernières étant :

- La classe Game étant le « moteur » du jeu, elle va venir instancier toutes les autres classes du jeu. Elle contiendra par exemple les listes de salles, de sorties, d'unités, d'items etc.. Elle viendra faire aussi le pont entre les interactions possibles et le joueur.

- La classe Place va quant à elle faire office de salle qui contiendra une liste de sorties, d'unités, d'items. Elle permettra aussi de connaître ses salles voisines, de savoir si le joueur se trouve à l'intérieur ou non. Des éléments précieux à connaître pour les autres classes.

- La classe Exit représente les tous les types de portes du jeu. Elle sera donc une classe mère permettant de créer d'autres types de portes grâce à l'héritage. On se retrouvera donc à avoir des portes standards qui peuvent s'ouvrir et se fermer sans clé, des portes à verrous qui auront besoin de clé spécifique pour l'ouvrir. Et une porte magique qui ne s'ouvrira qu'à l'aide d'items spécifique.

- La classe Unit est une classe mère abstraite, elle se charge de matérialiser tout les types d'unités que le jeu peu comporter. C'est-à-dire dans un premier temps la classe Player, mais aussi les classes « PNJ » (personnages non-joueurs) hériteront de la classe Unit. La classe NPC/PNJ viendra se diviser aussi en deux catégories, une classe ennemie et une classe neutre (jardinier). La classe Unit va contenir les informations que peuvent contenir une unité en temps réel, comme par exemple un nom, des points de vie, une bourse, un inventaire, etc.. Elle aura bien évidemment la possibilité de se battre grâce à une implémentation de l'interface « Attack » afin que par la suite toutes les classes filles puissent redéfinir la méthode d'attaque en fonction de leur spécificité.

Documentation développeur :

- Diagramme de classes :

- La classe Item du jeu va être un peu la consistance de ce dernier. Elle sera elle aussi une classe mère abstraite. Pour palier au fait qu'elle soit abstraite elle aura une relation avec une classe concrète du nom d'« Identity » pour permettre de contenir les informations d'un objet. Une particularité avec la classe mère Item et qu'elle est en agrégation avec un item du nom de « Chest » lui permettant donc d'être à la fois un objet à part entière, mais aussi un contenant dans un coffre. La classe item va se diviser en deux catégories elle aussi, les items standards et les armes (classe Weapon).

Notes :
Dans la conception du diagramme de classes, nous avons aussi fait appel à différentes interfaces comme le « Printable », « Attack » , « Talk ». La première interface nous permet d'avoir un affichage sur des classes comme Unit, Place, Exit et Item. L'interface Attack sert à toutes les unités pour l'aspect du combat dans le jeu. Et l'interface Talk permet aux ennemies de pouvoir parler au joueur. À côté de cela, nous avons intégré aussi un système d'énumération permettant de créer un mécanisme de rareté sur les armes ainsi que sur les monstres. Ce qui implique donc qu'avec une arme de rareté inférieur à celui du monstre, on a peu de chances de pouvoir le battre. Nous avions eu l'idée aussi de pouvoir créer un personnage (le jardinier/gardener) faisant office de « marchand » afin d'augmenter les mécanismes du jeu. Concrètement, si le joueur décide de communiquer avec le jardinier du château et non pas de l'attaquer directement, alors il aura la possibilité de pouvoir faire des emplettes auprès de lui.

En parlant de mécanisme, on a prit la décision de rentrer certains items vraiment vitaux dans la progression du jeu, par exemple, on a un système de lumière grâce à des torches ou des bougies qui permet de voir ce qu'il y a dans le château en pleine nuit. Pour que ces deux items aient un minimum d'intérêt dans un jeu en mode texte, c'est qu'elles peuvent avoir un impact réel sur les actions du joueur. Dans notre cas, on pourrait faire en sorte que sans l'un des deux objets cités on ne puisse plus regarder ce qu'il se trouve dans la salle (coffres/objets divers/monstres/etc..). Concrètement, sans ces objets, il ne pourrait plus avoir des actions du style : LOOK, ATTACK, TAKE. Il ne lui resterait que la possibilité de retourner en arrière (GO salle d'avant) car il connaît normalement de tête où se trouve la porte derrière lui ou USE afin de recharger sa lampe torche ou de rallumer sa bougie. Ce qui fait que sans lumière, il se retrouve à la merci des monstres du château et est dans l'incapacité d'avancer dans le jeu.

Documentation développeur :

- Diagramme d’états :

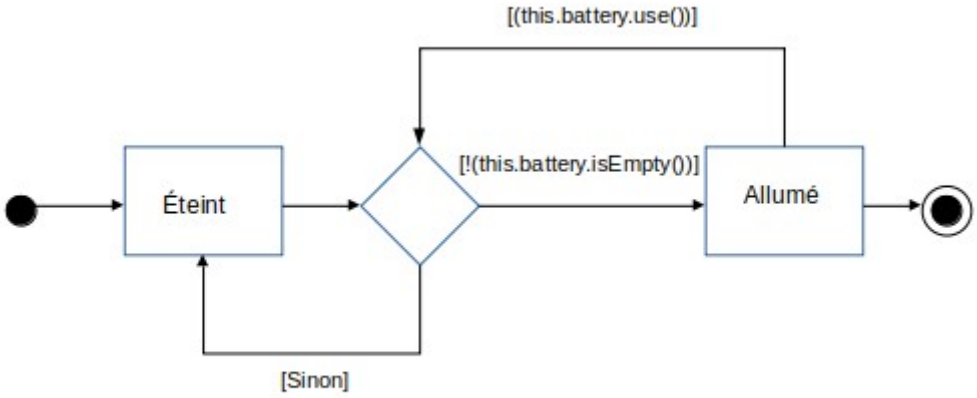
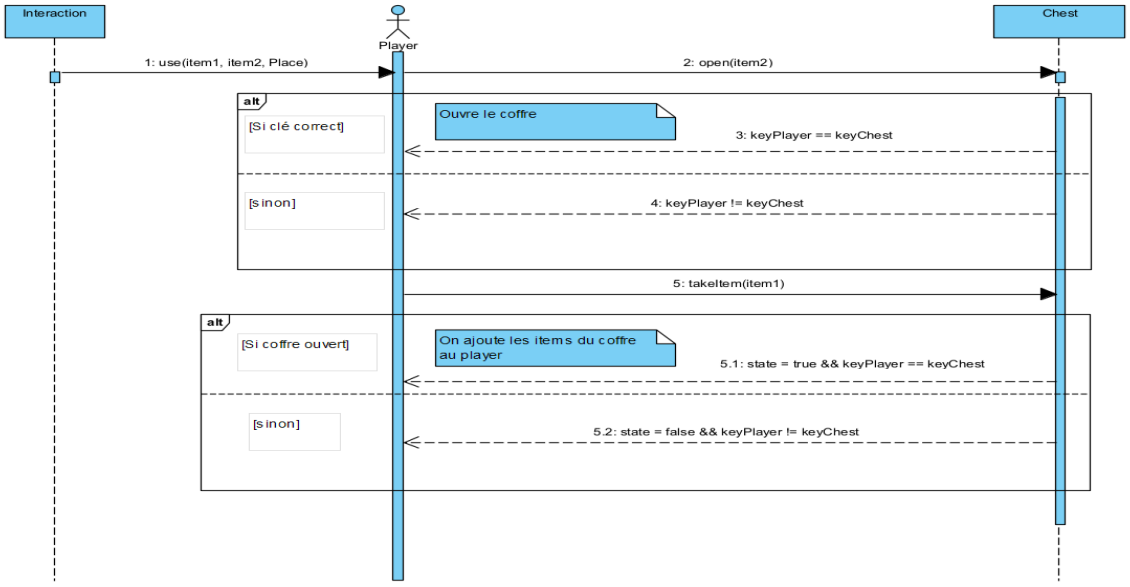


Diagramme d’états du Flashlight

Exemple des états possibles de la lampe torche

Explication : À l’état initial, la lampe torche est éteinte, si l’on l’allume et que l’on a de la batterie pour alors on passe à l’état d’allumer et on consomme une charge de batterie. Sinon on reste à l’état éteint. Ce fonctionnement-là est important pour nous, car comme dit précedemment, sans lumières on ne peut avancer dans le jeu.

- Diagramme de séquences :



Organisation du travail de groupe :

De par notre nombre réduit pour réaliser ce projet, nous avons pu nous répartir convenablement le nombre important de tâches à faire pour construire le projet. Nous avons tous les deux contribué à part égale à la conception UML du jeu ainsi que des caractéristique que l'on voulait chacun voir mis dans le jeu.

Au sujet de la répartition du développement des classes, c'est un peu compliqué de préciser qui s'est retrouvé sur quelles classes, car étant donné le nombre conséquent de classes que l'on a dans notre projet, on a dû repasser plusieurs fois sur différentes parties de l'autre. Mais on pourrait dire qu'initialement Guillaume s'est chargé de créer les salles, les interactions ainsi que la classe mère Game. De mon côté, je me suis occupé des items, des portes ainsi que des unités. Puis l'on essayé de compléter des parties manquantes dans des classes quand on les apercevait.

Cela dit la cohésion de notre groupe était très bonne on savait ce que l'un l'autre voulait lors du développement et on tendait pour le faire. On s'est énormément servis de GitHub pour pouvoir constituer notre projet, ainsi des applications comme Discord et Google Doc pour échanger des idées et en conserver des preuves pour le développement. On s'est bien évidemment servi de Visual Paradigm pour faire notre conception UML (que ce soit pour le diagramme de classes, d'états et de séquences). Et on s'était contraint au maximum à respecter lors du développement les diagrammes que l'on avait faits au préalable.

Hélas, le jeu était beaucoup trop ambitieux pour le nombre qu'on était à le construire ainsi que le temps qui nous a fait défaut pour le débugger entièrement et le finir proprement. Cela nous a réellement attristé de ne pas avoir pu le finir convenablement avec l'énergie que l'on a investie dedans et le fait que l'on a voulu intégrer le plus de notions possibles vues dans le programme de la POO. Nous espérons que cela ne portera pas trop préjudice au rendu global du projet.

Remerciements