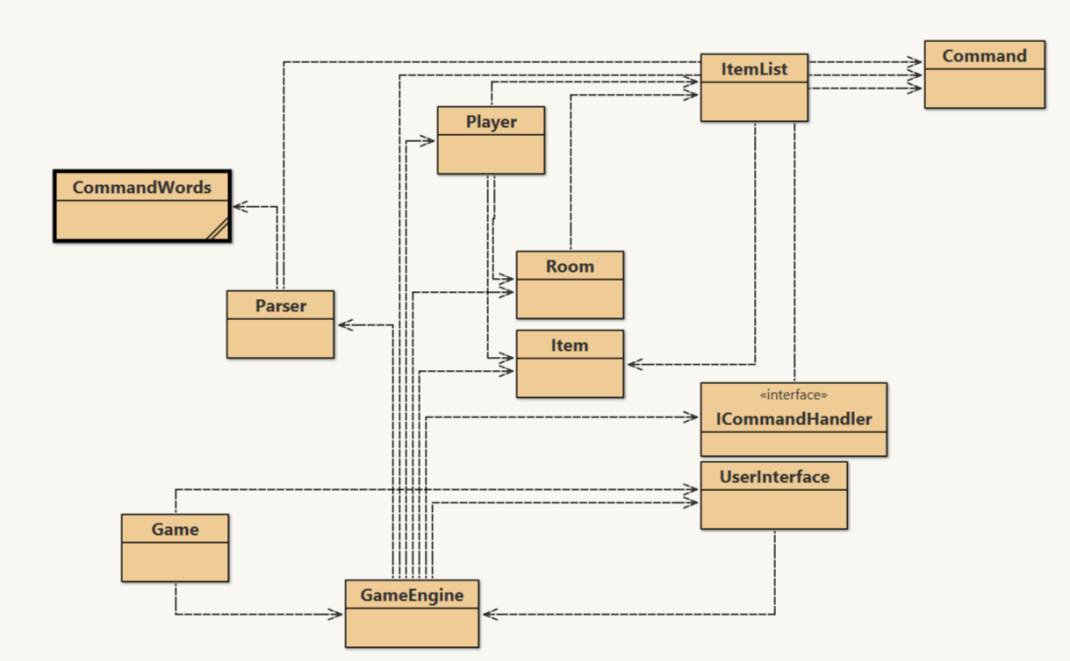
Rapport de Projet Zuul



I.A) L'Auteur

Le jeu est réalisé par Hugo Deniau en 12C.

I.B) Le Thème

Le jeu se nomme 'Lost in space' et s'articule autour de la phrase 'Dans un vaisseau spatial accidenté, un prisonnier rebelle doit s'enfuir.'

I.C) Le Synopsis

1) Le joueur démarre l'aventure dans une cellule, une panne de courant dans le vaisseau lui permet d'ouvrir la porte pour passer dans le couloir. L'aventure commence.
2) Le personnage choisit un sens pour explorer le couloir, d'un coté le couloir est bloqué par une porte fermée à clef (porte #1). De l'autre coté le couloir continue et débouche sur deux salles, une cafétaria et un laboratoire.
3) La cafétaria semble avoir été le siège d'important combats, les murs sont en effet marqués d'impacts de tirs encore chaud. Vous ne trouvez rien / vous trouvez un peu de nourriture, mais remarquez tout de même que la seule autre porte a été barricadée.
4) Vous entrez dans le laboratoire, il est désert mais vous voyez des animaux inconnus flottant dans ce qui ressemble à des aquariums. Vous observez un curieux appareil sur la paillasse. Il ressemble à une pince mesurant près d'un mettre, et une lueur orangée en émane. Un des documents que vous apercevez vous apprend qu'il s'agit d'un tournevis, vous pourrez surement démonter des trucs avec (voir porte #1)

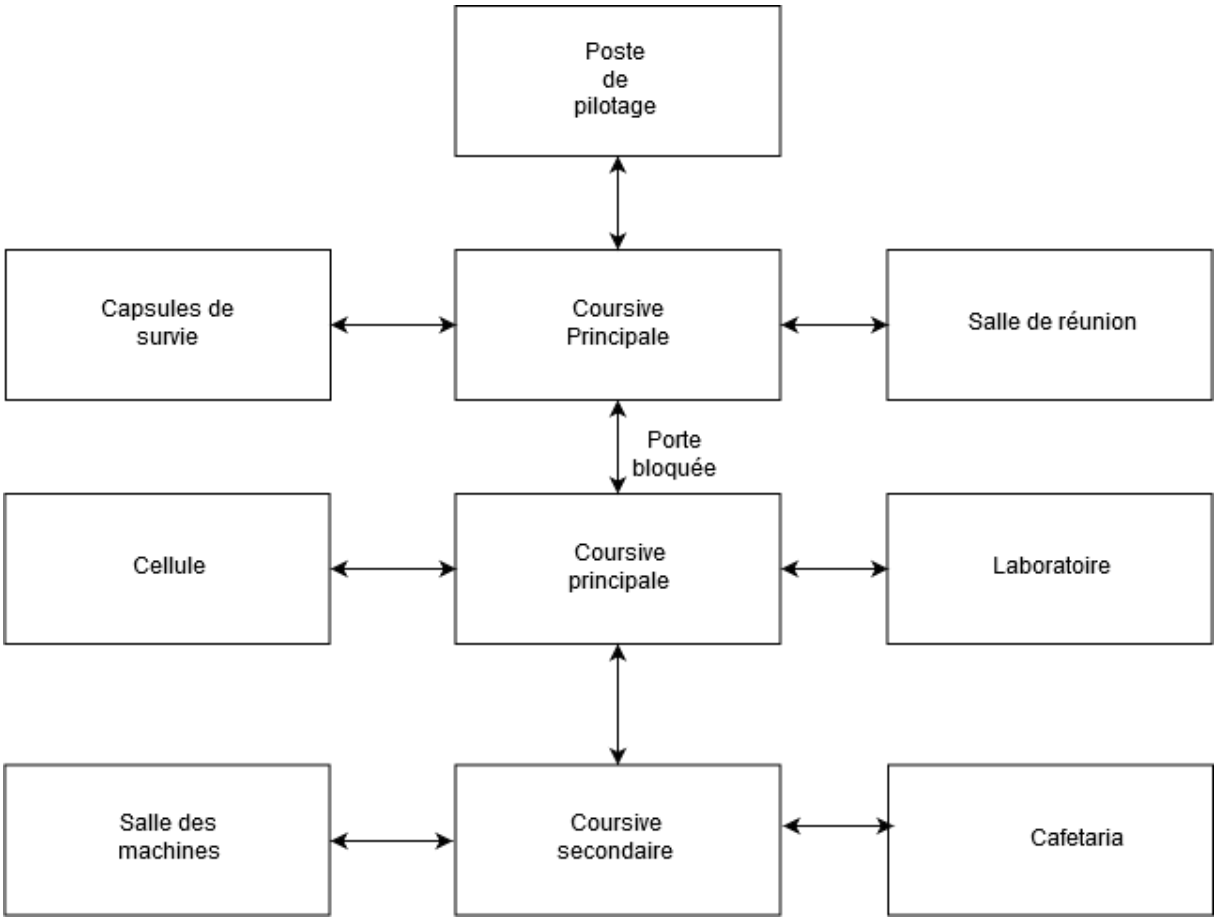
5) Après avoir dégagé la caisse, vous pénétrez dans un couloir et plusieurs portes se présentent à vous (le poste de pilotage, la soute qui est un déplacement vertical, et les capsules survie)
6) Vous pouvez enfin voir une échappatoire : il reste une capsule de survie amarré. Sans un regard derrière vous, vous entrez dans l'habitacle, et vous retrouvez propulsés hors du vaisseau, vous êtes en sécurité.

Situations gagnantes : Le joueurs s'échappe du vaisseau (éventuellement dans le temps imparti, pour ajouter un certain challenge au joueur).

Situations perdantes : Le joueur ne sort pas du vaisseau dans le temps imparti et meurt.

Enigmes : Ramasser un objet pour déplacer ouvrir la porte.

I.D) Plan des lieux



II) Exercices

7.5) Avant d'implémenter cette méthode, il y avait plusieurs fois les mêmes lignes de code a plusieurs endroits. Or la duplication de code est à proscrire et il est impératif e la limiter le plus possible.

7.6) J'ai remplacé les différents attributs présents dans la classe Room afin d'utiliser une HashMap qui associera les directions aux salles correspondantes. J'ai également rajouté une méthode pour récupérer une sortie en fonction de son nom.

7.7) Cela permet de séparer les différentes parties de l'application, notamment la parie qui gère les salles et la partie qui gère l'affichage du jeu.

7.10.2) La Javadoc ne liste que les méthodes publiques, donc il y a plusieurs méthodes invisibles dans la Javadoc.

7.15) Les méthodes *eat* et *look* ont étés rajoutées.

7.18) La méthode *getCommandList* a été rajoutée avec les différents changements que cela implique (même si cela ne change pas comment l'application se comporte).

7.18.2) Certaines méthodes ont été changées pour utiliser le StringBuilder .

7.18.5) Une HashMap a été ajoutée pour avoir accès aux objets de type Room après leur création.

7.18.6) L'interface a été ajoutée, avec les nombreux changements que cela inclus. La plupart des fonctionnalités de Game ont été déplacés dans le GameEngine

7.18.7) addActionListener permet d'ajouter un objet pour écouter les évènements se produisant sur le composant en question en lui passant en paramètre un objet implémentant l'interface ActionListener. Celle-ci possède une méthode actionPerformed qui sera invoquée quand un événement se produira.

7.18.8) Plusieurs boutons ont été ajoutés à la fenêtre. Pour les aligner, ils sont placés dans un JPanel avec un BoxLayout d'axe vertical.

7.20) La class Item a été ajoutée et une salle peut contenir un item.

7.20.1) La classe Item possède un getter pour lire sa description. Celle-ci est utilisée par la méthode Room#getLongDescription pour afficher l'item contenu dans cette salle. Cette chaine de caractère est récupérée par le GameEngine et envoyé dans UserInterface pour l'affichage.

7.22) et 7.22.1) L'attribut pour stocker un item dans une Room est maintenant de type Set<Item>, ce qui permet de stocker un nombre presque infini d'items sans autoriser de mettre deux fois le même item dans la salle. Les items sont affichés lorsque le joueur entre dans la salle.

7.22.2) Des items ont été ajoutés.

7.23) La commande back a été ajoutée.

7.24) et 7.25) La commande back ne permet que de revenir d'une salle en arrière, ce qui n'est pas optimal.

7.26) La commande back utilise maintenant un stack, c'est-à-dire une collection LIFO (last in last out) ce qui permet d'utiliser la commande back plusieurs fois et de revenir dans les salles précédentes.

7.26.1) Les deux Javadoc (userdoc et progdoc) ont été générées. Elles sont maintenant auto-générées par Gitlab grâce a l'utilisation de l'intégration continue, qui génère et met en ligne la documentation automatiquement (voir le fichier .gitlab-ci.yml disponible sur le dépôt gitlab https://perso.esiee.fr/~deniauh/)

7.27) Les test permettraient de savoir si le jeu fonctionne correctement (e.g. tester les fonctionnalités essentielles comme le déplacement, les items, etc)

7.28) et 7.28.2) La commande test et les fichiers associés ont été créés.

7.29) L'objet Player a été ajouté. Il stocke actuellement la salle actuelle et les salles précédentes ou le joueur a été.

7.30) Les méthodes take et drop ont été ajoutées.

7.31.1) La classe ItemList a été ajoutée et est utilisée dans Room ainsi que dans Player pour le stockage de leurs items respectifs.

7.32) Le poid max a été ajouté. Un joueur ne peut pas porter un total de plus de 10 unités de poids.

7.33) La command items a été ajoutée et permet d'observer le contenu du sac a dos de notre joueur.

7.34) Un Cookie Magique a été ajouté. J'ai également changé la command eat pour qu'elle prenne un nom d'item en argument , et, si possible (cad : si le joueur a l'objet et que c'est un objet mangeable) réalise l'action de manger cet item. Le magic cookie donne +10 en capacité de stockage d'item.

7.34.1) Les fichiers de test ont été étoffés pour pouvoir mieux tester le jeu.

7.34.2) Les deux javadocs sont générée automatiquement sur mon site : http://perso.esiee.fr/~deniauh

7.42) Un compteur de temps a été ajouté. Il est incrémenté a chaque déplacement du joueur. Lorsque cette valeur dépasse une valeur prédéfinie (ici 20 , laissant suffisamment de temps pour finir le jeux) un message de fin est affiché et le jeu s'arrête.

7.43) J'ai ajouté un passage depuis le laboratoire vers la cafeteria qui ne peut être pris que dans un seul sans. La commande back à été adaptée pour ne pas pouvoir tricher et passer à travers cette porte.

7.44) Le Beamer a été ajouté ainsi que deux commandes : 'use' qui permet de le charger, et 'fire ' qui permet de se téléporter. On peut se téléporter vers la salle ou il a été chargé.

7.45) La classe Door a été ajoutée. Elle contient plusieurs attributs, notamment les deux salles qu'elle relie ainsi que si elle est actuellement fermée a clef, ainsi que la clé qui permet de l'ouvrir. Pour ajouter ceci il a notamment fallu modifier la méthode GameEngine#createRooms pour que chaque porte relie bien les salles. Il y a donc maintenant une porte fermée entre les deux coursives principales.

7.46) Le salle de télétransportation a été ajouté en utilisant l'héritage. Celle salle est en effet une classe fille de Room qui redéfinit la méthode Room#getExit pour obtenir une salle aléatoire.

7.46.1) LA command alea a été ajoutée Un objet random se trouve dans la classe Game et est utilisé pour générer la salle du télétransporteur. La commande permet de changer la seed.

7.48) Etant donné que le jeu sa passe dans un vaisseau spatial abandonné, je n'ai pas crée des personnes mais des créature. J'ai donc crée plusieurs interface, d'abord Entity, qui sera implémentée par toutes les cr éature du vaisseau. Elle specifie un nom et une salle pour la créature. J'ai crée l'interface Talkable, qui

spécifie une methode getText pour les créature qui peuvent parler. J'ai ensuite crée une command talk pour parler aux créature, et ajouté un message informant si il y a des créature pouvant parler dans la salle.

7.49) J'ai créé une interface EntityMoving qui spécifie une méthode move et moveChance. Ces méthodes permettent à la créature de changer de salle, mais seulement avec un pourcentage de chance. Cela évite que les créatures se baladent trop fréquemment entre les salles.

7.50) 'public static int max(int a, int b)' de la classe Math

7.51) Ces méthodes sont statiques car on ne peut pas instancier la classe utilitaire java.lang.Math, elle possède un constructeur privé. On ne peut pas non plus l'étendre car elle est final.

7.52) Cela prend 0ms pour compter de 1 à 100

```
public static void main(String[] args) {
    long start = System.currentTimeMillis();
    for (int i = 0; i < 100; i++) {}
    System.out.println(System.currentTimeMillis()-start);
}
```

7.53) La méthode main existe. Je l'ai ajoutée il y a très longtemps d'ailleurs.

7.54) et 7.54.1) Le jeu peut être exécuté à partir du fichier .jar disponible sur mon site http://perso.esiee.fr/~deniauh

7.56)

1. Pouvez-vous appeler une méthode statique à partir d'une méthode d'instance ? Oui
2. Pouvez-vous appeler une méthode d'instance à partir d'une méthode statique ? Non
3. Pouvez-vous appeler une méthode statique à partir d'une méthode statique ? Oui

En effet, les méthodes statiques s'exécutant sans instance d'objet, elles ne peuvent pas appeler de méthodes d'instance. Le contraire est cependant vrai, même si on est dans une instance, on peut appeler une méthode statique.

7.57) Oui, il faut incrémenter un compteur statique dans la classe a chaque appel du constructeur.

```
class MyClass {

    private static int numberOfInstances = 0;

    public MyClass() {
        numberOfInstances++;
    }

    public static int numberOfInstances(){
        return numberOfInstances;
    }

}
```

7.58) Le fichier jar est généré et disponible sur mon site http://perso.esiee.fr/~deniauh

7.61) et 7.62) Les deux commandes ‘save’ et ‘load’ ont été ajoutées. Elles utilisent la sérialisation java pour sauvegarder l’ensemble du jeu. La plupart des classes implémentés donc Serializable. J’utilise un ObjectOutputStream pour transformer mon objet GameEngine en suite de byte envoyé dans un FileOutputStream qui sauvegarde le contenu du jeu dans un fichier. La commande ‘load’ possède une sécurité, elle ne peut charger que les fichiers qui ont été sauvegardés par le jeu. Il y a en effet une extension ‘.save’ pour les fichiers de sauvegarde, ainsi qu’un header de 6 bytes pour vérifier que le fichier est bien une sauvegarde.

J’ai également rajouté une JMenuBar permettant de charger et sauvegarder plus simplement la partie. J’utilise la classe JOptionPane présente dans Swing pour avoir le nom de la partie à sauvegarder. J’ai également une méthode permettant de regarder quelle sauvegardent existent dans le dossier de sauvegarde et propose à l’utilisateur d’en charger une à l’aide d’une liste de type ‘dropdown’.

7.63) Le jeu est fini. Il est possible de gagner en allant dans la salle des capsules de survie, ou de perde en mettant trop de temps (le vaisseau explose et la partie se termine).

J’ai créé un script qui génère la javadoc et la met en ligne a chaque commit grâce à l’intégration continue Gitlab. J’ai également créé un script Gitlab qui utilise Docker et Maven pour générer le fichier .jar et le mettre en ligne sur le site.