Rapport :

Welcome to the castle

I.A) Auteur

Abel DIDOUH

I.B) Thème (phrase-thème validée)

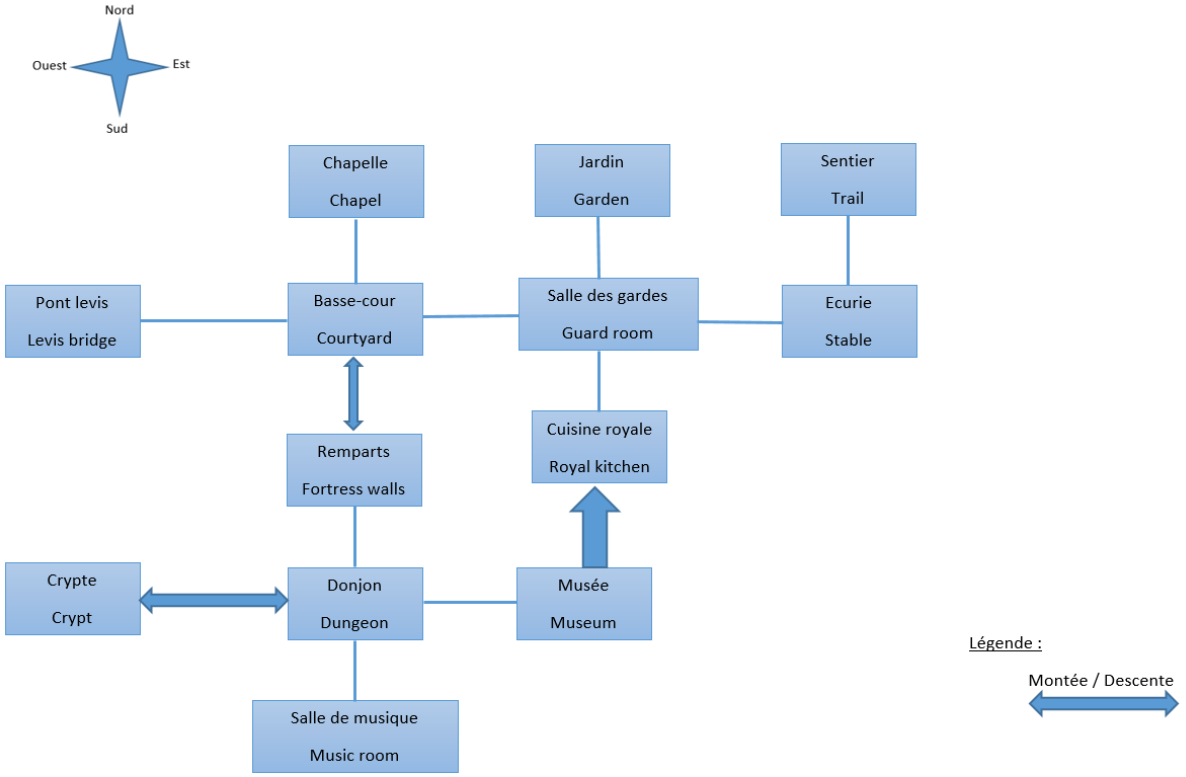
Dans un château-fort, un étudiant a pour mission de retrouver un parchemin.

I.C) Résumé du scénario (complet)

Un étudiant commence son aventure sur le pont-levis. Il doit avancer vers la basse-cour. Alors qu'il marche, il doit esquiver une attaque d'oie. L'étudiant rejoint le prêtre dans la chapelle. Il lui apprend qu'un parchemin est dissimulé dans le donjon. L'étudiant rejoint alors la basse-cour. Il doit retrouver la clé pour ouvrir le donjon. Il trouve alors la clé à côté d'une botte. Dans la salle des gardes, l'étudiant trouve des lunettes à vision nocturne. Mais échappe de peu à la ronde d'un garde suisse. Il prend ensuite une casserole dans les cuisines royales pour pouvoir bloquer la porte du donjon. Cet aventurier rencontre un cuisinier qui lui propose un gâteau. Il parcourt ensuite les remparts pour arriver dans une tour. Il croise un touriste possédant une carte du domaine. Ils échangent la carte contre un gâteau. L'étudiant marche alors vers le donjon. Monte les escaliers, ouvre la porte avec la clé trouvée dans la basse-cours. Il récupère la moitié du parchemin. L'autre moitié se trouve dans la crypte. L'étudiant descend tout l'escalier et se retrouve dans la crypte. Il récupère l'autre moitié. L'aventurier assemble les deux bouts, il lit : " Vous avez gagné !".

Abel DIDOUH

I.D) Plan (complet, avec indication de la partie "réduit" si exercice 7.3.3)



I.F) Détail des lieux, items, personnages

Lieux :

- Le pont levis : the levis bridge
- La basse cour : the courtyard
- La chapelle : the chapel
- La crypte : the crypt
- Les cuisines royales : the royal kitchens
- Le donjon : the dungeon
- Les ecuries : the stable
- Le musée : the museum
- Les remparts : the fortress walls
- La salle des gardes : the guard room
- La salle de musique : the music room
- Le jardin : the garden
- Le sentier : the trail

Items :

- Les lunettes nocturnes : the goggles
- La casserole : the pan
- Le ticket : the ticket
- La lampe : the light
- Les bottes : the boots
- Le cookie : the cookie
- La clé : the key
- Le gâteau : the cake
- La carte : the map
- le parchemin n°1 : the scroll 1
- le parchemin n°2 : the scroll 2

Personnages :

- l’oie
- le garde suisse
- le cuisinier
- le touriste
- le prêtre

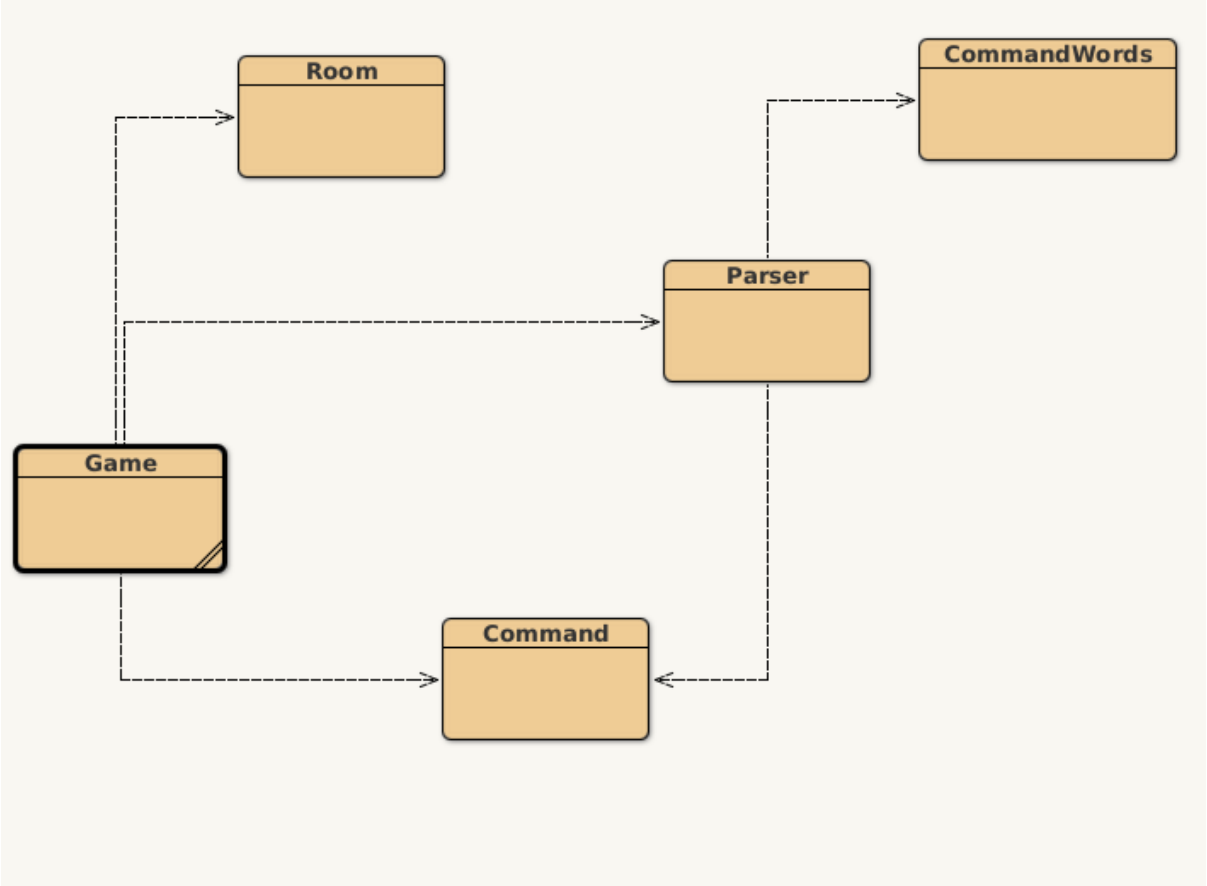
I.G) Situations gagnantes et perdantes

Le joueur gagne s'il trouve les deux bouts du parchemin.
Sinon il perd la partie.

I.H) Modification :

Exercice 7.5 :

Je récupère les 5 classes importantes du projet Zuul :
Game / Parser / Room / Command / CommandWords
Je sauvegarde ces 5 classes dans un nouveau projet.



Avez-vous compris pourquoi on vous demande de créer et d'utiliser cette procédure, et pourquoi la situation était "inacceptable" avant cette amélioration ?

Avant cette amélioration, il y avait une duplication de code.

Exercice 7.6 :

Je modifie la méthode goRoom(). J'effectue le remplacement comme indiqué dans le document ressource.

La version de `setExits()` proposée dans le livre teste si chaque paramètre est `null` avant de l'affecter à l'attribut correspondant, mais ce n'est pas indispensable. Comprenez-vous pourquoi ?
Ce n'est pas indispensable, car si c'est null il n'y a pas de sortie.

Exercice 7.7 :

Dans la classe Game, suppression des tests dans la méthode printLocationInfo().
J'effectue les tests dans la classe Room. Cela évite les problèmes d'encapsulation.

Comprenez-vous pourquoi il est logique de demander à Room de produire les informations sur ses sorties (et ne pas lui demander de les afficher) et pourquoi il est logique de demander à Game d'afficher ces informations (et ne pas lui demander de les produire) ?
On évite un problème d'encapsulation.

Exercice 7.8 :

Dans la classe Game, suppression de la partie b de la méthode goRoom.

Exercice 7.9 :

Je modifie la fonction getExitString(). Les sorties ne sont plus stockées de la même manière dans Room. J'implémente la nouvelle notion : HashMap.

N'y aurait-il pas une méthode inutile désormais dans la classe Room ? Par quoi a-t-elle été remplacée ?
SetExits() devient une méthode inutile dans la classe Room.
Elle a été remplacée par SetExit().

Exercice 7.16 :

Je modifie la classe processCommand.

Exercice 7.18.3 :

Je modifie toutes les images pour avoir des dimensions d'image uniforme 800px par 600px.

Exercice 7.18.6 :

J'étudie la documentation du projet zuul-with-images.
J'effectue les changements nécessaires dans la méthode createRooms().
Je transfère la plupart des méthodes se trouvant dans la classe Game vers la classe GameEngine. Je modifie la méthode processCommand() qui devient interpretCommand(). Cette méthode prend comme paramètre une chaîne de caractères.

Exercice 7.18.8 :

J'ajoute un bouton : quit. Lorsque le bouton est actionné, le programme appelle la méthode endGame() qui se trouve dans la classe GameEngine.

Exercice 7.20 :

J'étends mon projet d'aventure pour qu'une pièce puisse contenir un seul objet. Je sais qu'un objet possède un attribut nom et un attribut poids. Ainsi, j'ajoute des accesseurs pour récupérer les valeurs de ces attributs. Je crée également une méthode qui permet d'affecter un objet à une Room.

Exercice 7.21 :

Pour produire toutes les informations sur un objet présent dans une pièce, j'utilise les accesseurs sur les attributs qui m'intéressent. La classe qui doit produire la chaîne décrivant l'objet est la classe Item.
La classe qui doit l'afficher est la classe GameEngine. La classe GameEngine permet d'appeler this.aGui.println().

Exercice 7.21.1 :

J’améliore ma commande look. Ainsi, lorsque le joueur tape le nom d’un item après look, le programme affiche la description de l’objet.

Exercice 7.22 :

Je modifie mon projet pour qu'une salle puisse contenir un nombre quelconque d'éléments. Pour ce faire, je vais utiliser une hashmap. Cette collection permet de stocker des paires clé/valeur. Dans mon projet, cette collection sera de la forme (chaîne de caractère / Item).

Exercice 7.22.2 :

J'intègre les items de mon scénario.

Exercice 7.23 :

J'implémente une méthode : back. Lorsque l’utilisateur appelle la commande back, il revient dans la chambre précédente où il était.

Exercice 7.26 :

J'implémente la méthode : back. Ainsi, lorsque l’utilisateur appelle cette commande, il revient dans la chambre précédente. Je peux alors arriver dans la salle de départ si j’appelle assez de fois la commande.
Cette méthode utilise une pile. Lorsque l’on souhaite revenir, on dépile la pile. Si on avance dans les chambres, le programme empile la chambre actuelle sur la pile.

Exercice 7.26.1 :

Je génère les deux javadocs.
La première javadoc s’adresse à l’utilisateur d’une classe. Il ne voit que ce qui est public.
La seconde javadoc s’adresse au programmeur. Cette javadoc contient en plus tout ce qui est privé.

Exercice 7.28.1 :

Je crée une nouvelle commande : test.
Cette commande va exécuter successivement toutes les commandes lues dans le fichier texte.
Cette procédure accepte un paramètre de type chaîne de caractères (pNomFichier).
Pour pouvoir lire un fichier se trouvant dans un dossier à la racine du projet, j'utilise : this.getClass().getRessourceAsStream(...).

Exercice 7.28.2 :

Je crée trois fichiers de commande :
- court
- parcours
- exploration

Je place ces trois fichiers dans un dossier test.

Exercice 7.29 :

Je refactorise mon projet pour introduire une nouvelle classe Player.
Cette classe possède des attributs tels que le nom du personnage ou la salle actuelle où il se trouve.
Je migre toutes les méthodes présentes dans la classe GameEngine dans la classe Player.
La méthode goRoom possède 3 parties :
1) Gérer la direction c'est-à-dire tester si le second mot tapé existe.
2) La méthode goRoomPlayer(_Room_) présente dans la classe Player effectue le déplacement.
3) Afficher les informations de la nouvelle pièce courante.

Exercice 7.30 :

J'ajoute les fonctionnalités : take et drop à la classe Player.
La fonctionnalité take permet de prendre un item se trouvant dans la salle où se trouve le joueur.
La fonctionnalité drop permet de déposer un item dans la salle où se trouve le joueur.
On implémente la procédure take ayant un paramètre de type chaîne de caractères (pItemString). Et une procédure drop sans paramètre.

Dans la procédure take(final String pItemString) :

Je crée une variable de type Item et je lui associe l'item qui se trouve dans la salle où se trouve le joueur.
Si cet item est différent de null c'est à dire qu'il existe alors j'ajoute à aItem l'item portant le nom passé en paramètre.

Dans la procédure drop() :

A l'aide de la procédure addItem(), j'ajoute l'objet courant aItem à la chambre courante aCurrentRoom.

Exercice 7.31 :

J'ajoute la fonctionnalité : porter plusieurs items à mon jeu Welcome to the Castle.
Cette fonctionalité permet au joueur de porter plusieurs items.
Dans la classe Player, on ajoute un attribut aItems qui utilise une hashmap. Ainsi, on associe une chaîne de caractère et un item. Dans la procédure take() on utilise la procédure put(_String_, _Item_) sur l'objet courant aItems pour ajouter un item. De la même manière, dans la procédure drop(), j'utilise la procédure remove(_String_) sur l'objet courant aItems pour supprimer un item.

Exercice 7.31.1 :

Je crée une nouvelle classe ItemList.
Cette classe permet de gérer une liste d'items et ainsi mutualiser la gestion des items qui se retrouvait dupliquée dans Room et Player.
Ainsi, cette modification du choix de stockage des items internes à Room et à Player ne devrait entraîner aucune modification du moteur du jeu.
On utilise comme collection une hashmap pour stocker les items.
La collection d'objets utilisée dans ItemList n'est pas censée pouvoir être manipulée depuis l'extérieur d'ItemList.
Deplus l'ItemList n'est pas censée pouvoir être manipulée depuis l'extérieur de Room ou de Player.
Ainsi, la classe ItemList contient un attribut privé : aItems.
aItems utilise une hashmap pour associer un nom et un item.
Dans la classe ItemList, il y a un constructeur par défaut.
J'ajoute à cette classe une procédure qui ajoute un item à l'attribut aItems. Cette procédure possède deux paramètres. De la même manière, j'ajoute une procédure ayant comme paramètre une chaîne de caractère qui supprime l'item associé au nom passé en paramètre.
La fonction getItemsString est ajoutée à la classe ItemList.

Exercice 7.32 :

J'ajoute une restriction qui permet au joueur de ne transporter des objets que jusqu'à un poids maximum spécifié. Le poids maximum qu'un joueur peut porter est un attribut du joueur. Dans la classe Player, on ajoute un attribut poids maximum : private int aMaxWeight; et un attribut poids actuel : private int aCurrentWeight; . Dans le constructeur Player, j'ajoute comme paramètre : final int pMaxWeight . J'affecte 0 à aCurrenWeight.

Exercice 7.33 :

J'implémente une commande : inventory qui affiche les items porté par le joueur et le poids total. Je modifie les classes GameEngine, CommandWords et la classe ItemList. J'implémente une fonction inventory dans la classe ItemList qui va construire une chaîne de

caractères comportant le nom des items. J'ajoute des conditions : si l'attribut aItems est vide alors on renvoie un message : il n'y a pas de d'objet dans l'inventaire.

Exercice 7.34 :

Je crée un nouvel objet : cookie. Je place cet objet dans la salle : vSalleCuisine. Je modifie la commande eat dans la classe GameEngine. Si le second mot est cookie alors j'appelle la procédure magicCookie qui se trouve dans la classe GameEngine. Cette procédure double le poids maximal de l'inventaire du joueur et supprime l'objet magique de la salle.

Exercice 7.34.1 :

Je met à jour les fichiers de test.
Je modifie les trois fichiers : court, exploration et parcours.

Exercice 7.34.2 :

Je met à jour les 2 javadoc du projet comme à l'exercice 7.26.1.

Exercice 7.42 :

J'ajoute une forme de limite de temps à mon jeu. Si une certaine tâche n'est pas achevée dans un temps donné, le joueur perd. Une limite de temps peut facilement être mise en place en comptant le nombre de mouvements ou le nombre de commandes saisies. Pour cela, je crée un attribut : aNumberOfAction (int) . Je crée un accesseur et un modificateur sur cet attribut. J'implémente une procédure timeLimit. Dans un premier temps, la fonction récupère puis stocke dans une variable locale le nombre d'action stocké dans l'attribut aNumberOfAction. La fonction teste si la variable locale est égale à un certain nombre. Si c'est le cas, la fonction appelle la procédure endGame(). Sinon on incrémente la variable locale puis on écrase

l’attribut aNumberOfAction à l’aide du modificateur setANumberOfAction().
La limite pour terminer le jeu est de 20 actions.

Exercice 7.42.2 :

Je me contente de l’interface graphique actuelle de mon jeu.

Exercice 7.43 :

J'implémente une trapdoor entre le musée (museum) et la cuisine (kitchen). Le sens de la trapdoor est : du musée vers la cuisine. Tout d’abord, je modifie les sorties de la room cuisine pour créer la trapdoor. Ensuite, j’ajoute à la classe Room une méthode isExit(Room) qui renvoie vrai ou faux selon que la Room passée en paramètre est une des sorties de la pièce ou pas. Dans cette méthode je crée une liste locale à la méthode qui enregistre les noms des rooms en sortie. A l’aide de la méthode contains() avec comme paramètre le nom de la room où se trouve le joueur :

```
vListNameRoom.contains(pRoom.getNameRoom());
```

J’obtiens une valeur booléenne que j’utilise dans la classe Player. Ainsi, si la méthode isExit(.) renvoie false, la pile se vide, place l’ancienne salle comme nouvelle salle à l’aide de la méthode setCurrentRoom(.) et ajoute l’ancienne salle au sommet de la pile.

Exercice 7.43.1 :

Je met à jour les 2 javadoc du projet comme à l'exercice 7.26.1.

A SAVOIR EXPLIQUER :

- Stack

Une pile est une structure de données.

- push()

La méthode push() ajoute un objet au sommet de la pile.

- pop()

La méthode pop() supprime un objet se trouvant au sommet de la pile puis renvoie l'objet supprimé.

- peek()

La méthode peek() retourne l'objet se trouvant au sommet de la pile. Cette méthode ne supprime pas l'objet.

- lecture simple de fichiers de texte

Pour lire un fichier texte on utilise la classe Scanner. On crée un objet de type Scanner.

- File

La classe File permet de gérer les chemins d'accès pour accéder au fichier.

- Scanner (différence avec le 7.2.1 ?)

La classe Scanner permet de lire des fichiers textes.

- hasNextLine()

La méthode hasNextLine() est un itérateur. Cette méthode permet de savoir s' il y a une prochaine ligne.

- nextLine()

La méthode nextLine() est un itérateur.

IV. Déclaration obligatoire anti-plagiat **(*)**

Je déclare ne pas avoir copié de ligne de code provenant d'une tierce personne.
Je me suis uniquement basé sur les documents ressources.
Les images proviennent d'Internet.

Sources Images :

https://lepetitnancay.com/wp-content/uploads/2017/11/26chateau-blois%C2%A9CDT41-dsourice-Poster-300x200.jpg

https://upload.wikimedia.org/wikipedia/commons/e/e9/La_Chapelle_d%27Abondance._Int%C3%A9rieur_de_la_chapelle_Notre_Dame_de_Compassion._2015-06-20.JPG

https://upload.wikimedia.org/wikipedia/commons/1/1f/Bayeux_cathedrale_Notre-Dame_crypte.jpg

https://www.lelude.com/wp-content/uploads/2021/06/chateau-du-lude-cuisines.jpg

https://1.bp.blogspot.com/-eHSpDMfCv6Q/Xkm7CqWI_wI/AAAAAAAArUg/sS4x2vMemJom9fGNdSj6jCxf1etRDaANACLcBGAsYHQ/s1600/800px-Septmonts_%2528int%25C3%25A9rieur_donjon%2529_salle_sup%25C3%25A9rieure_6231.jpg

https://www.macommune.info/wp-content/uploads/2022/06/unnamed-1.jpg

https://www.gruissan-mediterranee.com/wp-content/uploads/wpetourisme/Chateau-remparts-Carcassonne-remparts-nord-C.Jeanjean-CMN--8-.JPG

Abel DIDOUH

https://www.paj-mag.fr/wp-content/uploads/sites/3/2021/02/salle-des-gardes-82-tarn-et-garonne-chateau-de-goudourville_bgvmf_2-1200x914.jpg

https://encrypted-tbn1.gstatic.com/images?q=tbn:ANd9GcSB4JBiQ54Fyde6nr1i0LLzRnCJAB0S9bzDej1i0Fjwhu6zmDEf

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRqdP7bdiWj48KWoJAQvkCwdrOYTYFLZ1VrS2vXTtrY76AfdjCa

https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcS3URJ0-zpqJPpg3h4uVzPcWA2PvwMg-FX09uYnso276Gj49xOa

https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcRygJAhChtDZ6Jk6PzoBdsOyc_RkaAbSHyV6M4cNL9C__2V7WgS

https://encrypted-tbn2.gstatic.com/images?q=tbn:ANd9GcR5TdSvnKnLTNW88RztaHHW_FH29flbJGZA-5qhht8P42iJLPkU