Claire BESANCON
Xavier BAUBET
Arthur BERTRAND-BERNARD

L3 informatique 2024/2025
UFR SFA – Université de Poitiers
03/12/2024

Rapport de projet

UE Programmation orientée objet et interfaces homme-machine 1

Table des matières

1- Documentation utilisateur

The underground adventure est un jeu qui se déroule dans une ville souterraine ou le but est de sortir de celle-ci. Pour cela, il vous faudra trouver trois parties de clés cachées dans la ville pour ensuite faire la clé complète qui vous servira à remonter à la surface. Au lancement du jeu, vous allez apparaître dans le quartier principal, vous y trouverez les principaux lieux de vie tels que la taverne, la boutique et d'autre ! Dans certains de ces lieux, vous pourrez trouver des personnages, les aidants qui vous donnerons des indices si vous leur parlez et les revendeurs qui vous donneront un item contre un ou plusieurs autres items.

Ce jeu est basé sur la logique de commande textuelle, voici toutes les commandes disponibles :

-HELP : la commande « help » vous sera probablement très utile en jeu, car elle vous permet de réafficher toutes les commandes disponibles à n'importe quel moment dans votre partie.

-LOOK : la commande « look » sert à regarder autour de vous, cette commande vous donnera toutes les informations de la pièce où vous vous trouvez, comme les sorties disponibles, la liste des items et des personnages présents dans la pièce.

-LOOK [objet] : à la différence de la commande « look » sans arguments, celle-ci prend le nom de l'item en argument et vous donnera une description de l'objet demandé (il faut bien évidemment que l'item que vous souhaitez regarder soit présent dans la pièce ou dans votre sac.).

-SPEAK : « speak » sert tout simplement à lancer le dialogue avec les différents personnages présents du jeu, attention pour cette commande, il faut vérifier au préalable que vous n'êtes pas seul dans la pièce (avec la commande look).

-TAKE [objet] : la commande « take » est très simple, elle vous sert à prendre les différents items présents dans les pièces, cependant pour récupérer certains items, il vous faudra un item particulier en votre possession (Si c'est le cas, un message apparaîtra vous disant : pour récupérer cet item, il vous faut cet item.).

-INVENTORY : la commande « inventory » sert à vous afficher la liste des items actuellement présent dans votre sac, bien évidemment il vous faudra récupérer le sac pour utiliser cette commande.

-EAT : « eat » vous sert à manger, elle est très utile quand vous avez perdu de la vie, cette commande permet de consommer un item de type food et ainsi de restaurer vos points de vie. Pour cela, il faudra trouver de la nourriture dans la carte.

-READ : La commande « read » permet de lire les livres, ceux-ci peuvent, vous donnez de précieux indices, un livre est requis pour pouvoir utiliser cette commande.

-STATE : la commande « state » vous permet de vérifier votre niveau de vie actuel.

-TIME : la commande « time » vous permet de regarder le temps qu'il vous reste dans votre partie si vous avez lancé celle-ci en mode compte à rebours.

-QUIT : cette commande vous permet de quitter le jeu. Attention, il n'y a aucune sauvegarde de votre partie, si vous quittez le jeu, vous recommencerez au tout début !

Passons maintenant aux commandes qui vous permettront de vous déplacer dans la carte.

Il y a plusieurs types de sortie :
- Les sorties « libres » ou aucune action n'est nécessaire pour accéder à la pièce suivante, il vous suffira d'utiliser la commande :
    - GO [nom de la sortie] : pour avoir toutes les sorties disponibles, il vous suffira d'entrer la commande LOOK avant la commande GO celle-ci vous donnera toutes les sorties disponibles ainsi que leur nom. ATTENTION, il faut impérativement écrire le nom de la sortie avec le nom donné par la commande LOOK (majuscule et minuscule comprise) sinon le jeu ne pourra pas trouver la pièce recherchée !

Exemple : GO Square west

- Les sorties verrouillées avec un code, ces sorties nécessite plusieurs actions, tout d'abord utiliser la commande GO celle-ci vous retournera un message si la sortie n'est pas ouverte, il faudra donc utiliser la commande :
    - UNLOCK [nom de la sotie] qui vous demandera ensuite de rentrer un code pour pouvoirs la déverrouiller.

Ensuite, ouvrez la porte avec la commande :
    - OPEN [nom de la sortie]

Et enfin, vous pourrez changer de pièce avec la commande GO décrite juste au-dessus.

- Les sorties verrouillées avec une clé, celles-ci ont exactement le même principe que les sorties à code à la différence que ces sorties-là n’ont pas besoin de code, mais d’une clé pour pouvoir sortir.

Concernant la fin du jeu plusieurs fins sont possibles, tout d’abord en utilisant la commande QUIT, cela vous fera quitter le jeu. Le jeu se terminera aussi dès que vous mourez et enfin si vous réussissez à rejoindre la surface, cela signifie que vous aurez gagné le jeu.

2- Documentation développeur

Diagramme de Classes UML

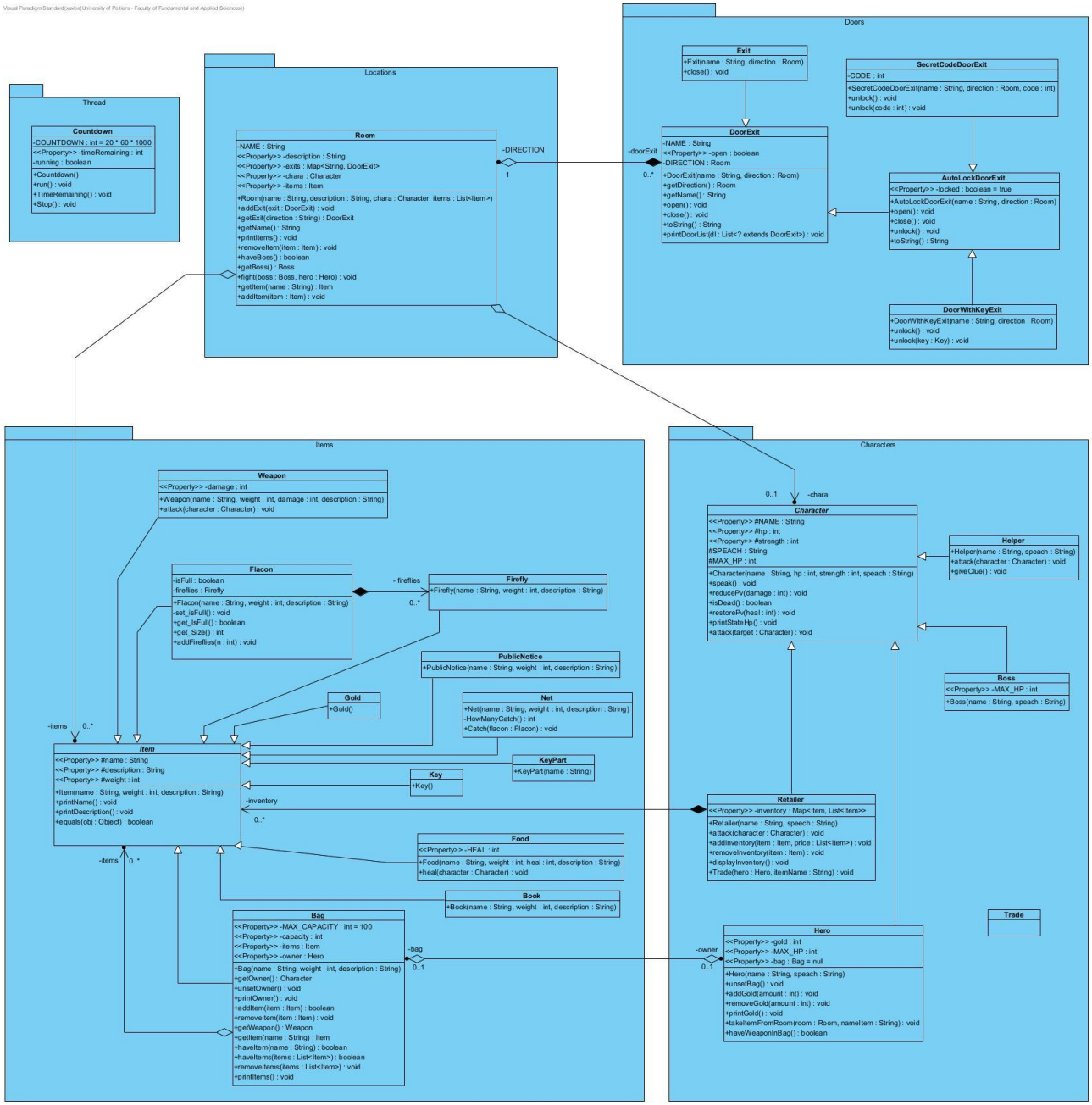

Figure 1 - Diagramme de classes UML

Légende du Diagramme de Classes UML

- Package (Rectangle avec onglet) :
    - Regroupe plusieurs classes.
    - Les packages sont utilisés pour organiser le modèle en sous-systèmes ou fonctionnalités.

- Classe (Rectangle) :
    - Représente une entité ou un objet dans le système.
    - Divisée en trois sections : nom de la classe, attributs et méthodes.

- Héritage (Flèche blanche avec triangle plein) :
    - Indique une relation d'héritage entre une classe parent et une classe enfant.
    - La flèche pointe vers la classe parent.

- Association unidirectionnelle (Flèche simple) :
    - Indique une relation où une classe connaît l'existence d'une autre classe et peut interagir avec elle.
    - La flèche pointe vers la classe cible (la classe avec laquelle elle interagit).

- Agrégation (Ligne avec losange vide) :
    - Indique une relation d'agréation où une classe est constituée de plusieurs instances d'une autre classe.
    - Le losange vide est du côté de l'agrégat (le tout/conteneur).

- Composition (Ligne avec losange plein) :
    - Indique une relation de composition plus forte où une classe est strictement constituée de plusieurs instances d'une autre classe.
    -Le losange plein est du côté du composite (le tout/conteneur).

- Définitions des Multiplicités :
    - 1 : Exactement une instance.
    - 0..1 : Zéro ou une instance.
    - 0..* : Zéro ou plusieurs instances.
    - 1..* : Une ou plusieurs instances.

Choix de conception pour la classe abstraite Item

La classe Item a été définie comme une classe abstraite pour plusieurs raisons :

1. Centralisation des attributs et méthodes communes : Item regroupe des méthodes, qui permettent d'éviter la duplication de code dans les sous-classes, et des attributs communs à tous les types d'objets. Par exemple, chaque item doit avoir un poids parce qu'il est utile à la classe bag qui doit gérer une capacité maximale.

2. Encapsulation : En définissant des méthodes génériques dans Item, nous pouvons fournir des comportements de base que les sous-classes peuvent utiliser ou surcharger. Par exemple, la méthode getWeight() a une implémentation commune, alors que la méthode addFirefly par exemple est une méthode qui est seulement utilisé par la classe Flacon.

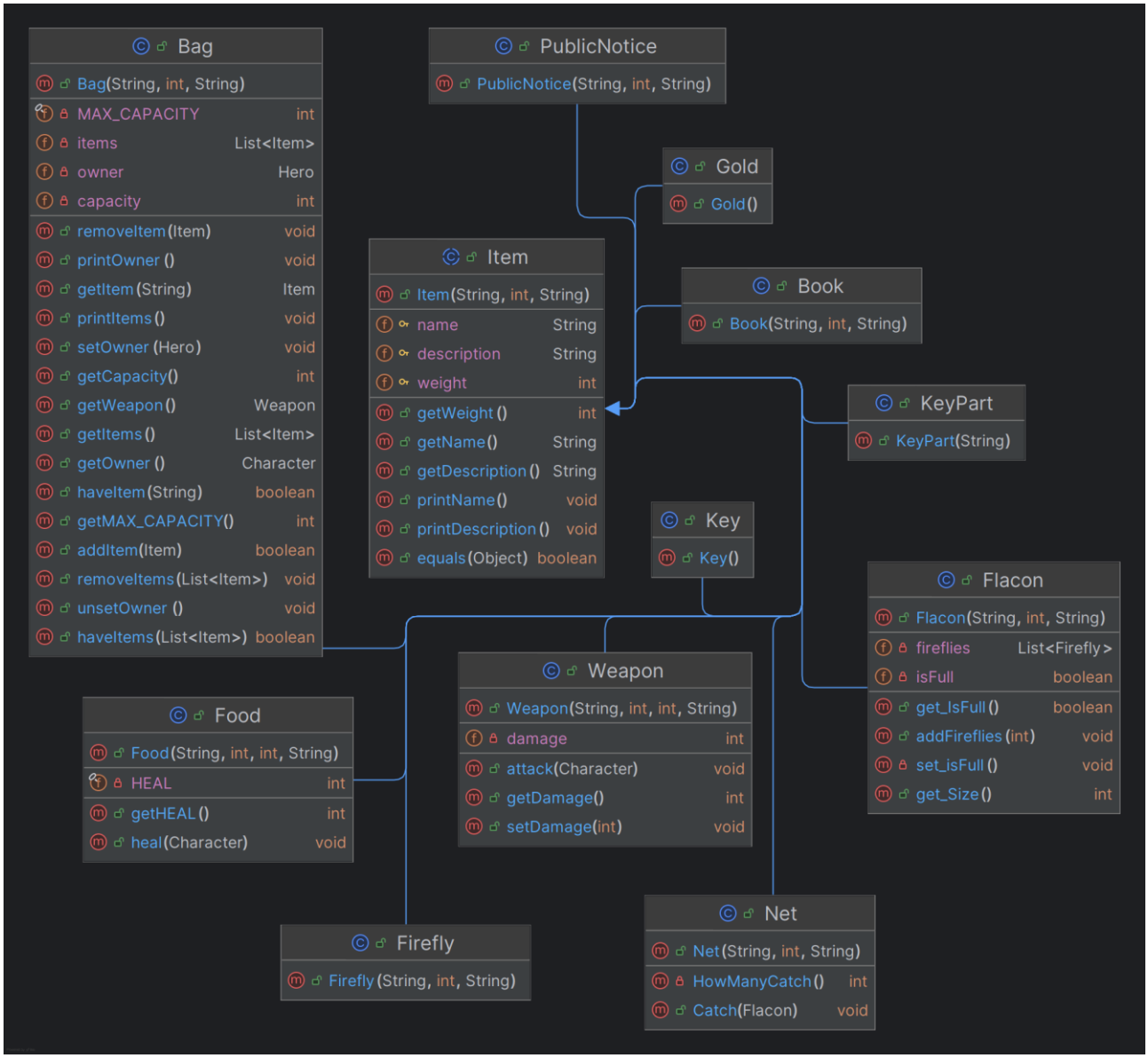

Figure 2 - Package Items

Choix de conception pour la classe Bag

La classe Bag est conçue pour contenir des items, et repose sur :

1. Agrégation : Bag possède une relation d'agrégation avec Item, ce qui signifie qu'un sac peut contenir plusieurs items. Cette relation reflète la réalité où un sac peut contenir divers objets sans être dépendant de leur type spécifique.

2. Limitation de capacité : L'attribut MAX_CAPACITY impose une limite au nombre d'objets que le sac peut contenir, ajoutant ainsi une contrainte de jeu qui peut influencer les décisions du joueur.

3. Gestion des items : En centralisant la gestion des items dans un sac, il devient plus facile de gérer l'inventaire du personnage, y compris l'ajout, la suppression, et l'accès aux items.

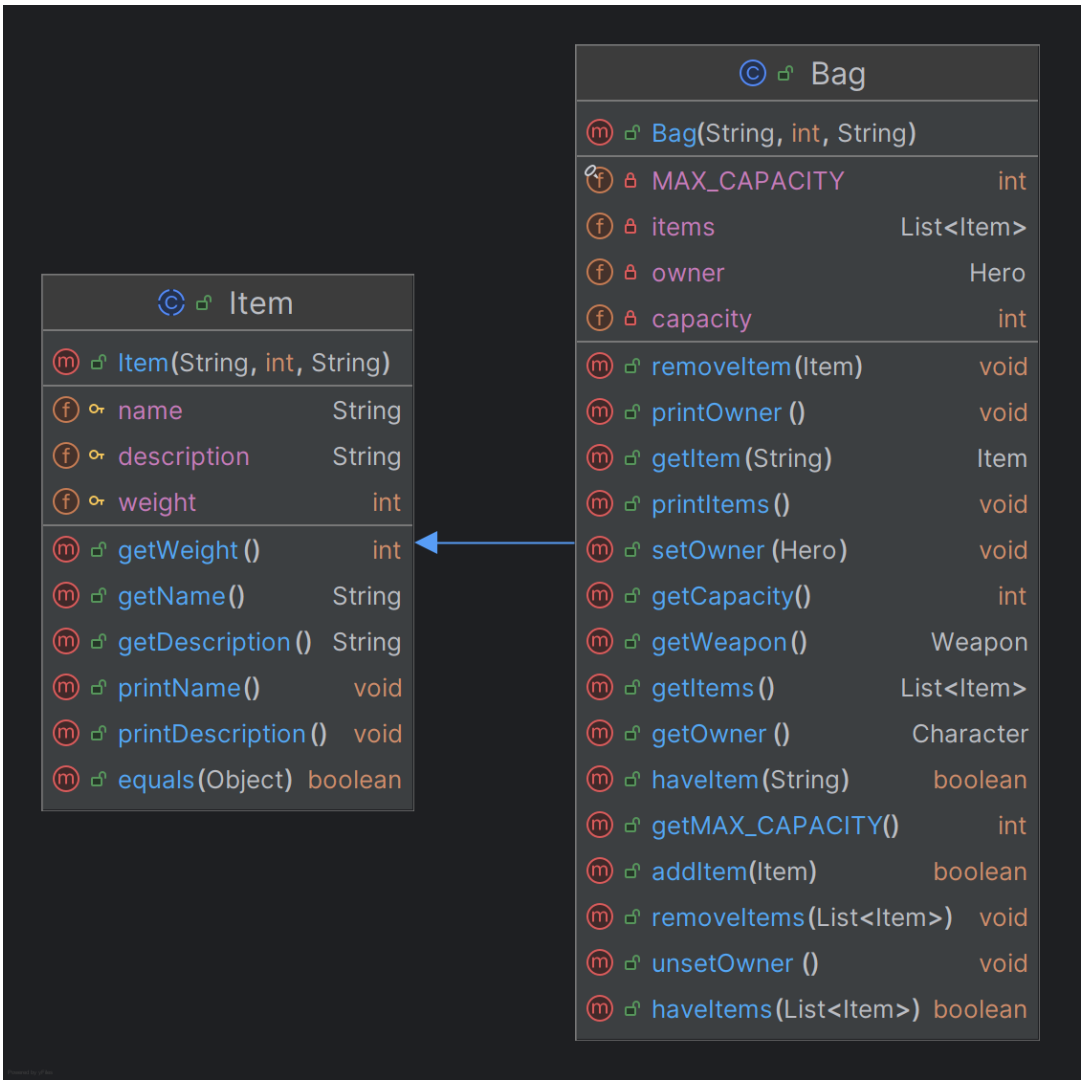

Figure 3 - Classe bag

Choix de conception pour la relation entre Room et Door

La relation entre Room et Exit est essentielle pour les connexions entre différentes pièces dans un jeu :

1. Navigation entre les pièces : La relation bidirectionnelle permet à chaque Room de connaître ses portes (Map<String, DoorExit>), et à chaque Door de connaître les pièces qu'elle connecte (direction).

2. Gestion des accès : En associant des portes aux pièces, on peut facilement gérer l'accès et les restrictions entre les pièces. Par exemple, une DoorWithKeyExit nécessite une clé pour être ouverte.

3. Flexibilité : En utilisant une classe de base DoorExit avec des sous-classes comme DoorWithKeyExit, SecretCodeDoorExit, etc., il est facile d'étendre le système avec de nouveaux types de portes sans modifier la structure de base des pièces.

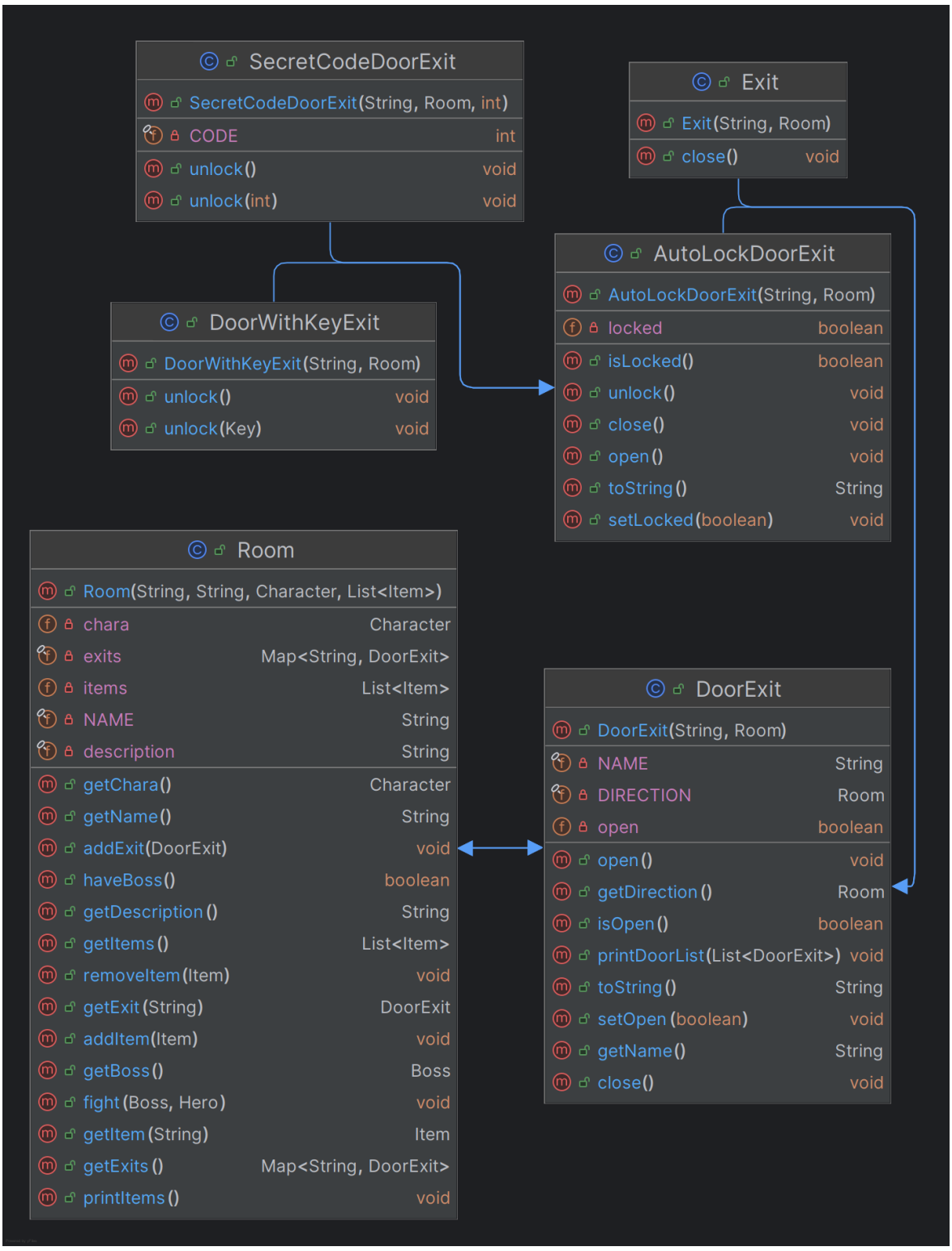

Figure 4 - Package Locations + Doors (liens)

Choix de conception pour les sous-classes de Character

Le choix de créer des sous-classes spécifiques pour Character est basé sur :

1. Spécialisation : Les sous-classes permettent de spécialiser le comportement et les attributs des différents types de personnages. Par exemple, un Hero possède la capacité de se rendre des points de vie.

2. Réutilisation : Les attributs et méthodes communs sont définis dans Character, ce qui facilite la réutilisation code. Par exemple, des méthodes comme réducePv() ou speak() sont implémentées dans Character et utilisées par toutes les sous-classes.

3. Flexibilité : En ajoutant de nouvelles sous-classes de Character, il est possible d'introduire de nouveaux types de personnages avec des comportements uniques sans affecter les autres parties du code.

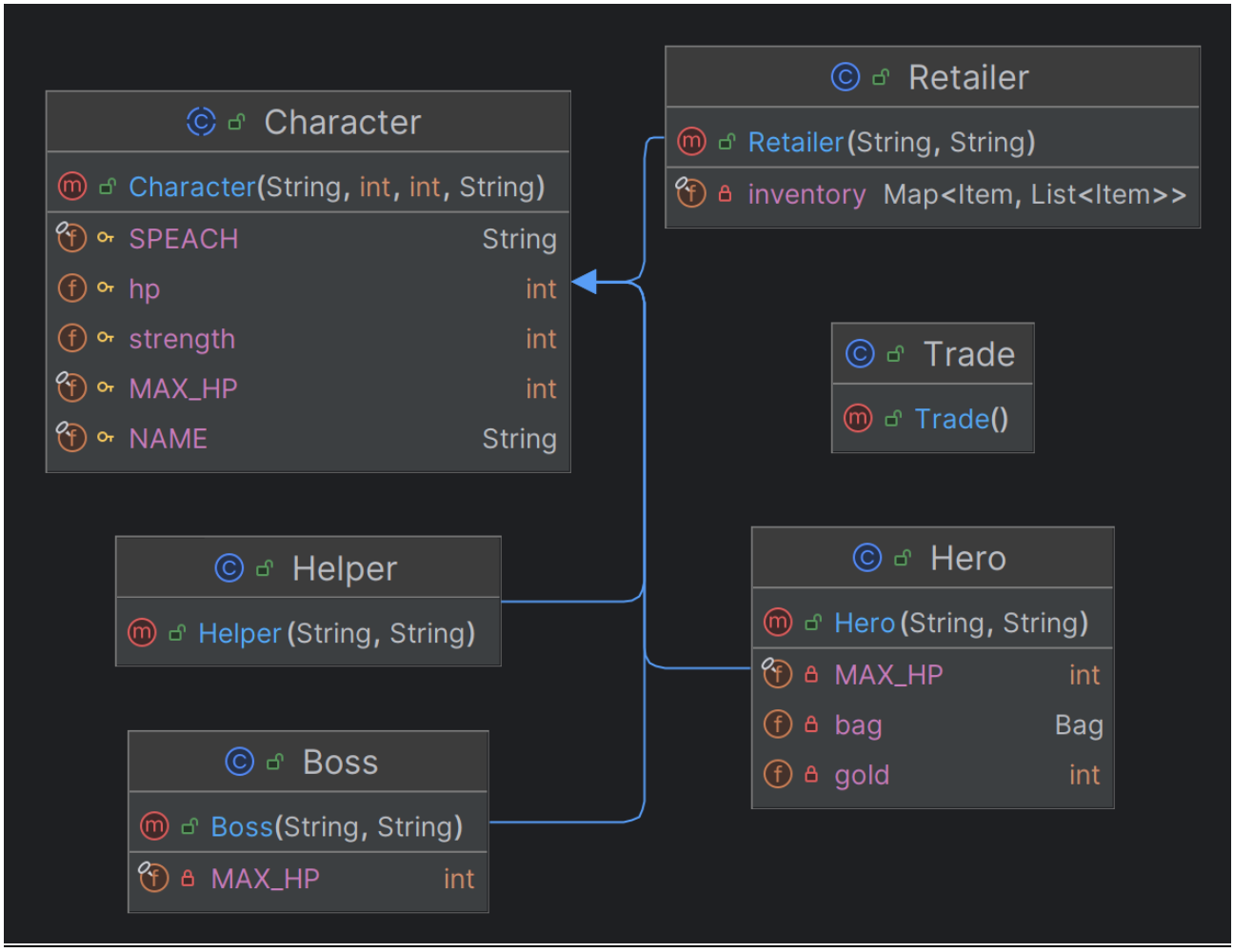

Figure 5 - Package Characters

Conception des commandes

Pour ajouter une commande au jeu il vous suffit d'ajouté une nouvelle condition à la boucle principale du jeu. Pour lire l'entré saisi par l'utilisateur vous devez utiliser la méthode equalsIgnoreCase si votre commande ne prend pas d'autre paramètres, sinon utiliser la méthode startsWithIgnoreCase qui prend en paramètre l'entré complète saisi par l'utilisateur ainsi que le début de la chaîne voulue.

Exemple d'utilisation :

if(command.equalsIgnoreCase("COMMANDE"){

*****corps de la condition*****

}

*Cet exemple est une nouvelle commande sans argument.

if(startWithIgnoreCase("COMMANDE "){

***** corps de la condition*****

}

*Cet exemple est une nouvelle commande qui prend un argument

Attention :

- pensez à ajouter un espace après la commande

- pensez à soustraire à la chaîne de caractère les premiers caractères correspondants à la commande : par exemple commande faisant 8 caractères plus 1(l'espace) = 9 caractères

if(startWithIgnoreCase("COMMANDE "){

String cmd = command.substring(9);

***** corps de la condition*****

}

Conception des tests

Le package Test de notre projet est dédié aux tests unitaires, qui ont été conçus pour vérifier la validité des constructeurs et des méthodes de nos classes principales. Chaque test unitaire se concentre sur une fonctionnalité spécifique, s'assurant que les différentes unités de code fonctionnent comme prévu de manière isolée. Pour garantir une couverture complète et précise, nous avons implémenté des tests pour les scénarios positifs et négatifs, en simulant diverses conditions d'utilisation.

Les tests sont structurés pour être exécutés de manière indépendante les uns des autres, permettant une détection rapide et efficace des régressions et des anomalies dans le code.

Composition :

1. Variables d’instances.
    - Exemple : *private Weapon pickaxe;*
2. Méthode de configuration :
    - Exemple :

    ```
    @Before
    public void setUp() {
    pickaxe = new Weapon("Pickaxe", 50, 50, "Desc pickaxe");
    }
    ```

3. Méthodes de test :
    - Exemple :

    ```
    @Test
    public void testFightWithPickaxe() {
       assertTrue(…);
    }
    ```

Ces tests sont exécutés en utilisant JUnit4, spécifiquement la version 4.13.1 (junit-4.13.1.jar).

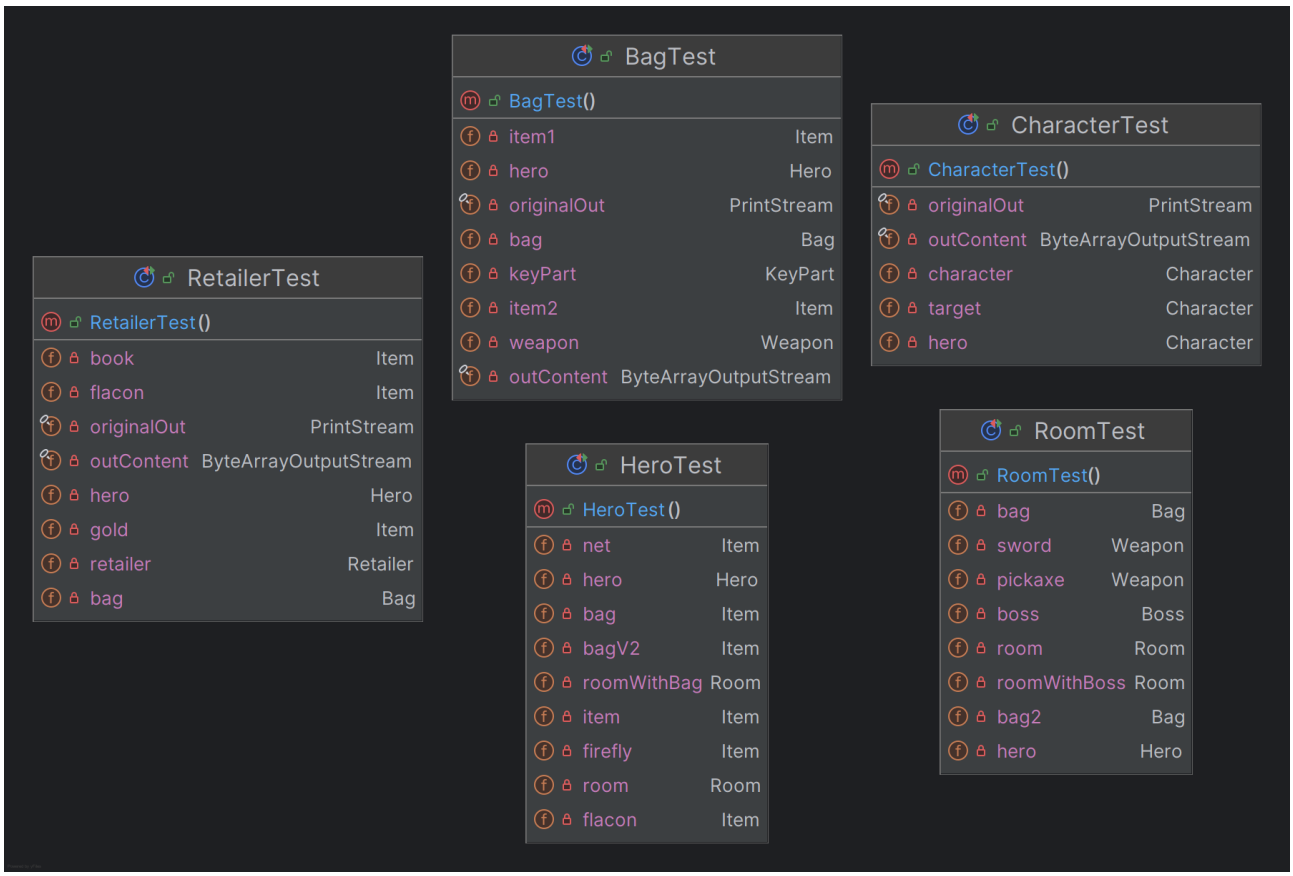

Figure 6 - Package Test

3- Organisation

Xavier :

- Réalisation de tout le package Item ainsi que le package Thread.
- Réalisation des fichiers de tests HeroTest et RoomTest
- Réalisation du README
- Réalisation du diagramme de classes

Arthur :

- Réalisation des packages Characters, Exits, Location ainsi que la boucle principale du jeu dans le fichier Main
- Réalisation des fichiers de tests BagTest, CharactersTest et RetailerTest
- Réalisation du rapport de projet

Claire :

- Réalisation de toute la partie description des items, des pièces, les différents discours des personnages ainsi que toutes les déclarations des items, pièces et exits .