Programmation Orienté Objet

L3 Informatique

LUNETEAU Thomas

VIALLE Charlie

BOUE Alexis

TABLES DES MATIÈRES

INTRODUCTION

Lors de ce projet de programmation orientée objet, nous avons eu 1 mois afin de réaliser un jeu textuel en java à la manière de “*Colossal cave Adventure*”. Il s’agit donc de faire un jeu d’aventure interactif où le programme décrit une situation par un affichage texte et où le joueur indique ce qu’il souhaite faire en saisissant du texte. Le programme analyse la phrase entrée par l’utilisateur et réagit à son tour en affichant la nouvelle situation, et ainsi de suite.

Nous avons choisi de faire un jeu d’aventure, nommé “Space and Pens”, se déroulant sur une planète étrangère où le but est de partir de la planète à la suite d' un crash sur celle-ci. Pour cela il faudra explorer la planète à la quête de trois objets magiques afin d’affronter le monstre final qui nous donne les objets nécessaires afin de partir de la planète. Le combat est une part importante de notre jeu étant donné qu’il permet d’obtenir 2 des 3 pièces et les dialogues permettent d’avoir la dernière. L’observation du décor permet de repérer et ramasser le réacteur du vaisseau nécessaire pour s'échapper de la planète.

1. Documentation utilisateur

Dans ce jeu, votre objectif est de s'échapper de la planète sur laquelle vous vous êtes crashé. Pour cela, vous affronterez de nombreux monstres dangereux, vous devrez les vaincre pour obtenir de l'argent afin d'acheter de meilleures armes et armures. Vous devrez explorer ce nouveau monde vous devrez trouver les 3 objets magiques nécessaire pour affronter celui qui possède tout ce dont vous avez besoin pour quitter cette planète.

a) Liste des commandes :

i) Hors combat

1) go

Entrez *go* suivi de la direction souhaitée pour vous déplacer à travers les différents tableaux. Les directions possibles sont *north*, *south*, *east* et *west* pour les déplacements classiques, *shop* pour entrer dans un magasin si il y en a un dans le tableau du joueur et *back* pour quitter le magasin.

2) talk

Entrez *talk* suivi du nom de la personne à qui vous voulez parlez .

3) help

Entrez *help* pour afficher en jeu toutes les commandes accessibles à ce moment.

4) look

Entrez *look* suivi de ce que vous voulez parmis les choix suivants, *here* pour savoir ce qu'il y a autour de vous, *inventory* pour connaître votre inventaire, *stat* pour voir les statistiques de votre personnage, *potion* pour connaître pour connaître le nombre de potions de chaque type que vous avez, *enemy* pour avoir la liste des ennemis affrontable sur ce tableau, *npc* pour avoir la liste des personne à qui vous pouvez parler sur ce tableau, *equipment* pour voir les objets que le joueur a comme équipement, *ground* pour voir la liste des objets ramassable sur le tableau, *money* pour savoir combien vous avez de pièce d'or dans le jeu, *shop* (si vous êtes dans un magasin) pour connaitre tout ce qu'il y a à acheter.
Si vous voulez voir la description d'un objet vous devez écrire *look item* puis le nom de l'objet.

5) take

Entrez *take* suivi de ce que vous voulez prendre au sol, vous pouvez savoir s' il y a des objets au sol avec la commande : *look ground* .

6) use

Entrez *use* suivi du nom de l'objet que vous voulez utiliser.
Si vous utilisez une potion (*health_potion, attack_potion, defense_potion, crit_potion*) alors vous la buverez. Une health_potion rend 25 points de vie tandis que les 3 autres potions ne peuvent être utilisées que en combat et doublent (lors d'un combat) respectivement l'attaque, la défense et le taux de critique(chance de faire deux fois les dégâts normaux en une attaque).
Si vous utilisez un objet alors vous l'équiperez mais vous ne pouvez avoir qu'une armure et une arme en même temps.Une armure augmente la défense tandis qu' une arme augmente l'attaque.

7) remove

Entrez *remove* suivi du nom de l'objet équiper afin de le déséquiper pour rajouter de la difficulté au jeu.

8) fight

Entrez *fight* sur un tableau qui contient des ennemis (look enemy) pour les affrontés.

9) buy (Utilisable que dans un magasin)

Entrez *buy* suivi du type de l'objet voulu (*item* pour un objet et *potion* pour une potion) puis de son nom (*look shop*)

10)sell (Utilisable que dans un magasin)

Entrez *sell* suivi du nom de l'objet que vous voulez vendre. Vous ne pourrez pas vendre des objets nécessaires à la complétion du jeu.

11)quit

Entrez *quit* pour quitter le jeu.

ii) En combat

1) attack

Entrez *attack* suivi du nom de l'ennemi viser (faite attention à bien entrer le nom).

2) defend

Entrez *defend* pour doubler votre défense jusqu'à votre prochain tour.

3) look

Entrez *look* suivi de ce que vous voulez parmis les choix suivants, *here* pour savoir ce qu'il y a autour de vous, *inventory* pour connaître votre inventaire, *stat* pour voir les statistiques de votre personnage, *potion* pour connaître pour connaître le nombre de potions de chaque type que vous avez, *enemy* pour avoir la liste des ennemis affrontable sur ce tableau, *npc* pour avoir la liste des personne à qui vous pouvez parler sur ce tableau,

equipment pour voir les objets que le joueur a comme équipement, *ground* pour voir la liste des objets ramassable sur le tableau, *money* pour savoir combien vous avez de pièce d'or dans le jeu.
Si vous voulez voir la description d'un objet vous devez écrire *look item* puis le nom de l'objet.
Si vous regardez pendant un combat alors vous ne pourrez ni attaquer ni vous défendre.

4) help

Entrez *help* pour afficher en jeu toutes les commandes accessibles à ce moment.

5) use

Entrez *use* suivi du nom de l'objet que vous voulez utiliser.
Si vous utilisez une potion (*health_potion, attack_potion, defense_potion, crit_potion*) alors vous la buverez. Une health_potion rend 25 points de vie tandis que les 3 autres potions ne peuvent être utilisées que en combat et doublent (lors d'un combat) respectivement l'attaque, la défense et le taux de critique(chance de faire deux fois les dégâts normaux en une attaque).
Si vous utilisez un objet alors vous l'équiperez mais vous ne pouvez avoir qu'une armure et une arme en même temps.Une armure augmente la défense tandis qu' une arme augmente l'attaque.

6) remove

Entrez *remove* suivi du nom de l'objet équiper afin de le déséquiper pour rajouter de la difficulté au jeu.

7) quit

Entrez *quit* pour quitter le jeu.

b) Fonctionnement d'un combat

Un combat vous oppose à un ou plusieurs ennemis. Un tour se déroule de cette façon, le joueur choisi et fait son action en voyant le nombre d'ennemis et leurs points de vie (mais pas leurs défense ni leurs attaque) puis tous les ennemis attaquent le joueur, les tours continues jusqu'à ce que tous les ennemis soit mort ou que le joueur n'ai plus de points de vie.
Les dégâts infligés sont la différence entre l'attaque de l'attaquant de la défense du défenseur. Si jamais l'attaque est inférieure à la défense alors c'est l'attaquant qui subit la différence entre l'attaque et la défense. Lors d'une attaque, il y a une faible chance que ce soit un coup critique qui multiplie l'attaque par deux.
Quand vous tuez un ennemi vous récupérez son argent et ses objets (s'il en avait).

c) Astuces

i) Comment s'équiper

Améliorer son équipement est très important pour cela, le magasin est votre meilleur ami, il possède des armes et des armures très puissantes si vous en avez les moyens. Le meilleur moyen d'obtenir de l'argent est par les combats mais vous pouvez aussi vendre vos anciens équipements ou des objets inutiles pour gagner plusieurs pièces d'or. Une légende raconte qu'un homme savant peut vous donner le pouvoir si vous arrivez à le trouver. Sa dernière phrase aurait été "Mon trésor ? Je vous le laisse si vous voulez, TROUVER LE ! je l'ai laissé quelque part dans ce monde."

ii) Gagner ses combats

L'attaque n'est pas toujours la bonne décision. La défense est généralement une bonne solution.

Utilisez vos potions d'attaque si vous savez que vous allez survivre à la prochaine attaque sinon utilisez une potion d'armure, celle-ci augmente autant votre défense que de se défendre ce qui permet d'avoir une défense impénétrable au cours du combat.

2. Documentation développeur.

a. Diagrammes

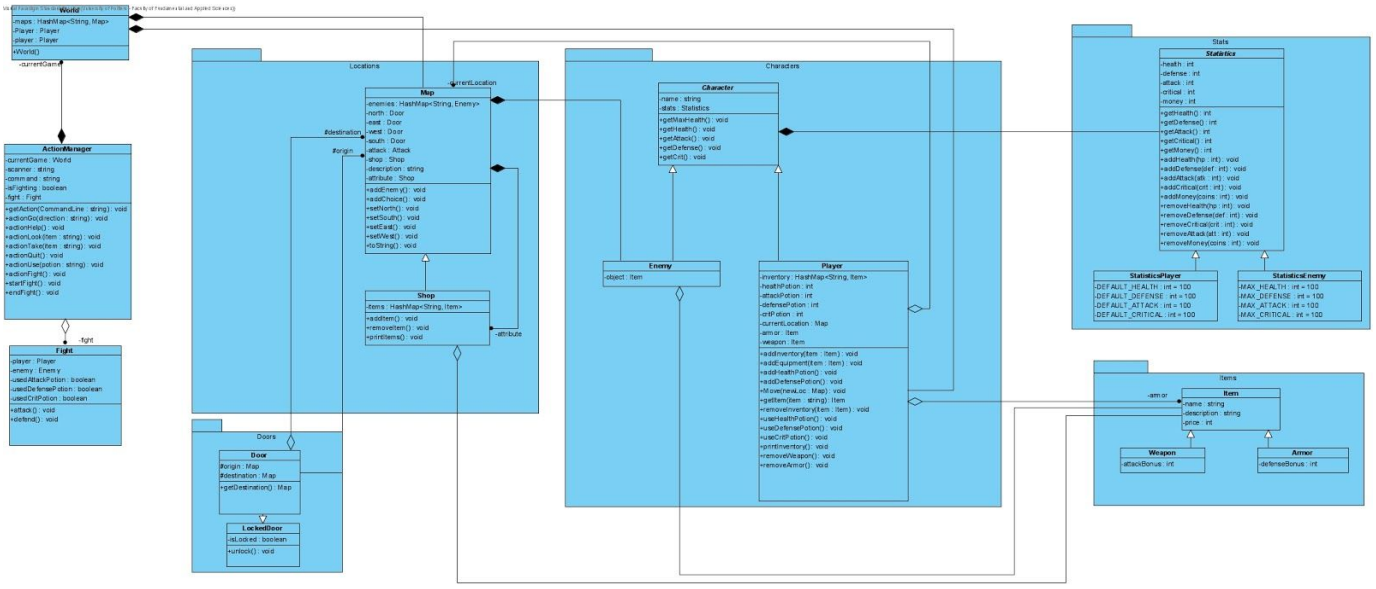


Diagramme UML initial

Lien pour télécharger Diagramme UML initial : https://imgur.com/a/gHCbvJJ

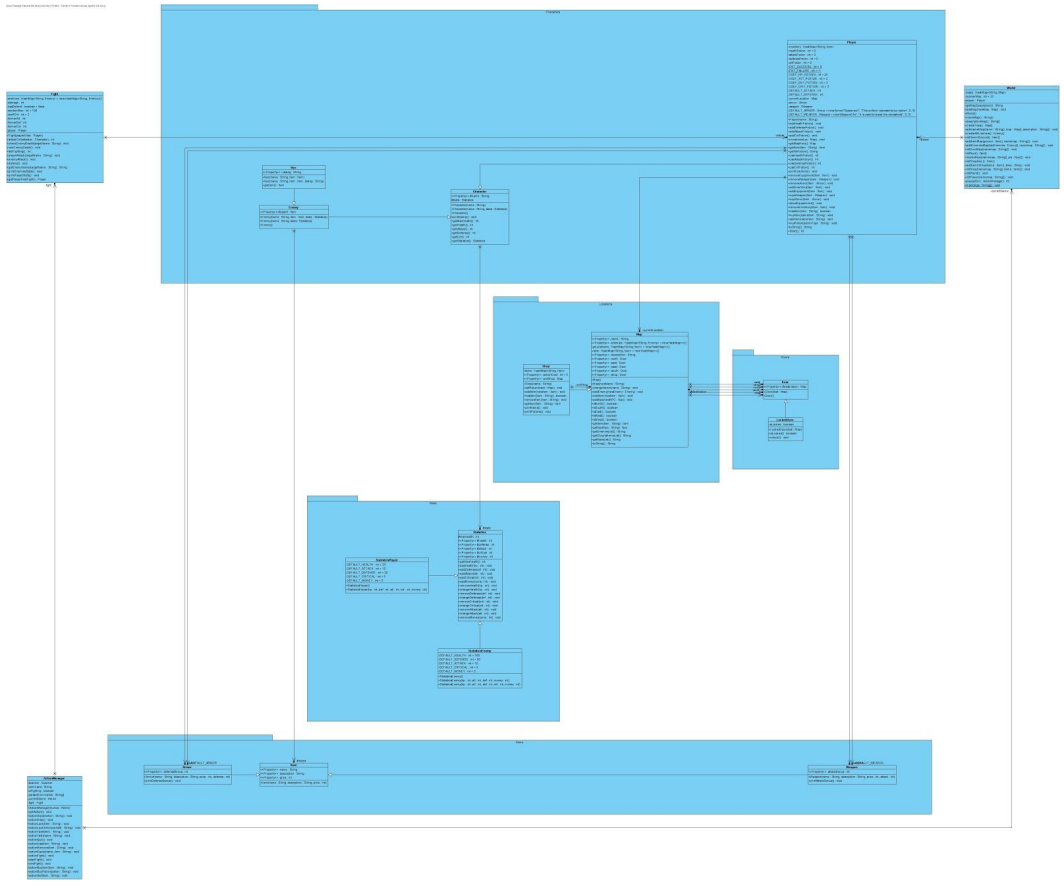
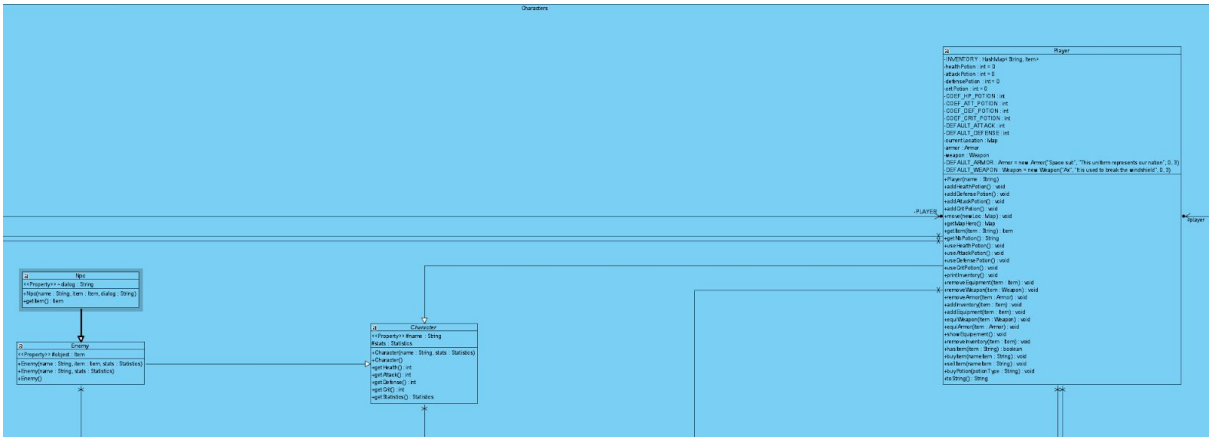


Diagramme UML final

Lien pour télécharger Diagramme UML final : https://imgur.com/a/xGlmbl2

Le diagramme de classe est resté en grande majorité inchangé au long du développement du jeu. Celui ci se compose de cinq packages :

Character :



Package contenant les classes des personnages du jeu.

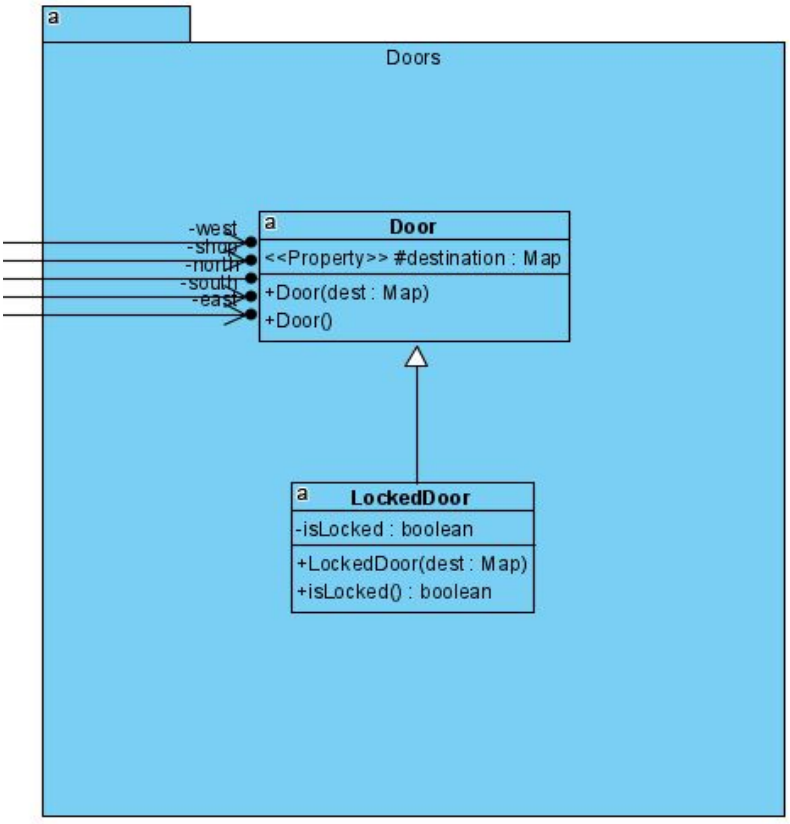
Character : Classe abstraite dont dérivent toutes les autres présentes dans le package. Elle contient les deux attributs élémentaires inhérents à chaque entité vivante du monde du jeu : un nom et des statistiques encapsulées dans un objet *Statistics*. Elle contient des accesseurs directs sur chaque statistique.

Enemy : Classe héritant de *Character*, celle-ci introduit un objet *Item*. Chaque ennemi peut donc disposer d'un objet quelconque.

Npc : Classe héritant de *Enemy*, celle-ci introduit une chaîne de caractère, le joueur est capable d'interagir avec les PNJ, ce qui fera afficher sur la console le dialogue stocké dans sa chaîne PNJ. Un PNJ est donc aussi potentiellement possesseur d'un objet qu'il peut léguer au joueur via la méthode getItem()

Player : Classe héritant de *Character*, c'est la plus grosse classe du package. Elle contient en plus un objet inventaire sous la forme d'une HashMap, des potions, une armure, une arme ainsi qu'une référence à la *Map* où celui-ci se trouve actuellement, ainsi le joueur sait toujours à quel endroit du monde il se trouve. Il dispose aussi des coefficients associés aux potions et d'une arme et d'une armure par défaut, inchangeables en jeu.

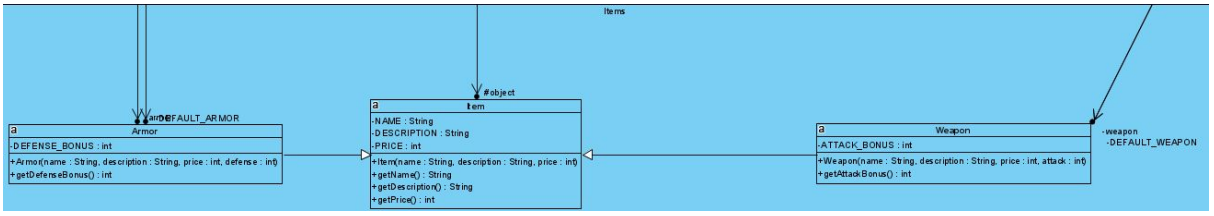
Doors :



Package contenant les classes des portes, joignant les *Map* et *Shop* entre eux.
Door : Classe contenant une référence vers une *Map* représentant la destination de la porte. Cette classe très simple contient évidemment une méthode get retournant la destination. Une fois la porte passée, il est impossible de revenir en arrière à moins que la *Map* suivante ait une autre *Door* permettant de revenir à la *Map* précédente.

LockedDoor : Classe héritant de *Door*, elle n'ajoute qu'un booléen isLocked permettant de verrouiller ou non une porte.

Items :



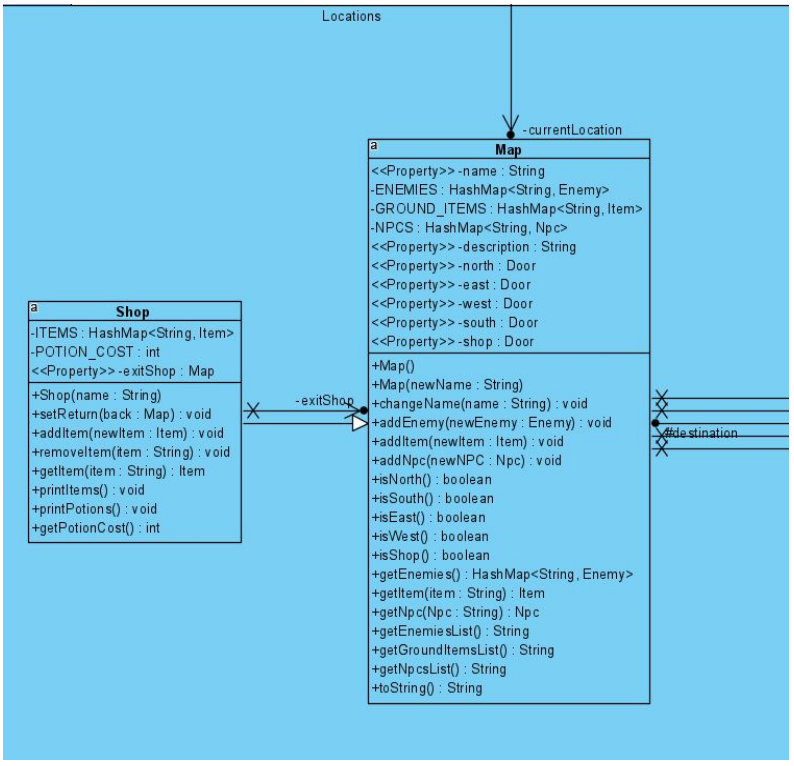
Package contenant les classes des objets du jeu.

Item : Classe des objets quelconque, ceux-ci sont définis par un nom, une description et un prix.

Armor : Classe des armures du jeu héritant de *Item,* elle apporte un bonus de défense, venant s'ajouter aux statistiques de défense du porteur lors des combats.

Weapon : Classe des armes du jeu héritant de *Item*, elle apporte un bonus d'attaque, venant s'ajouter aux statistiques de défense du porteur lors des combats.

Locations :

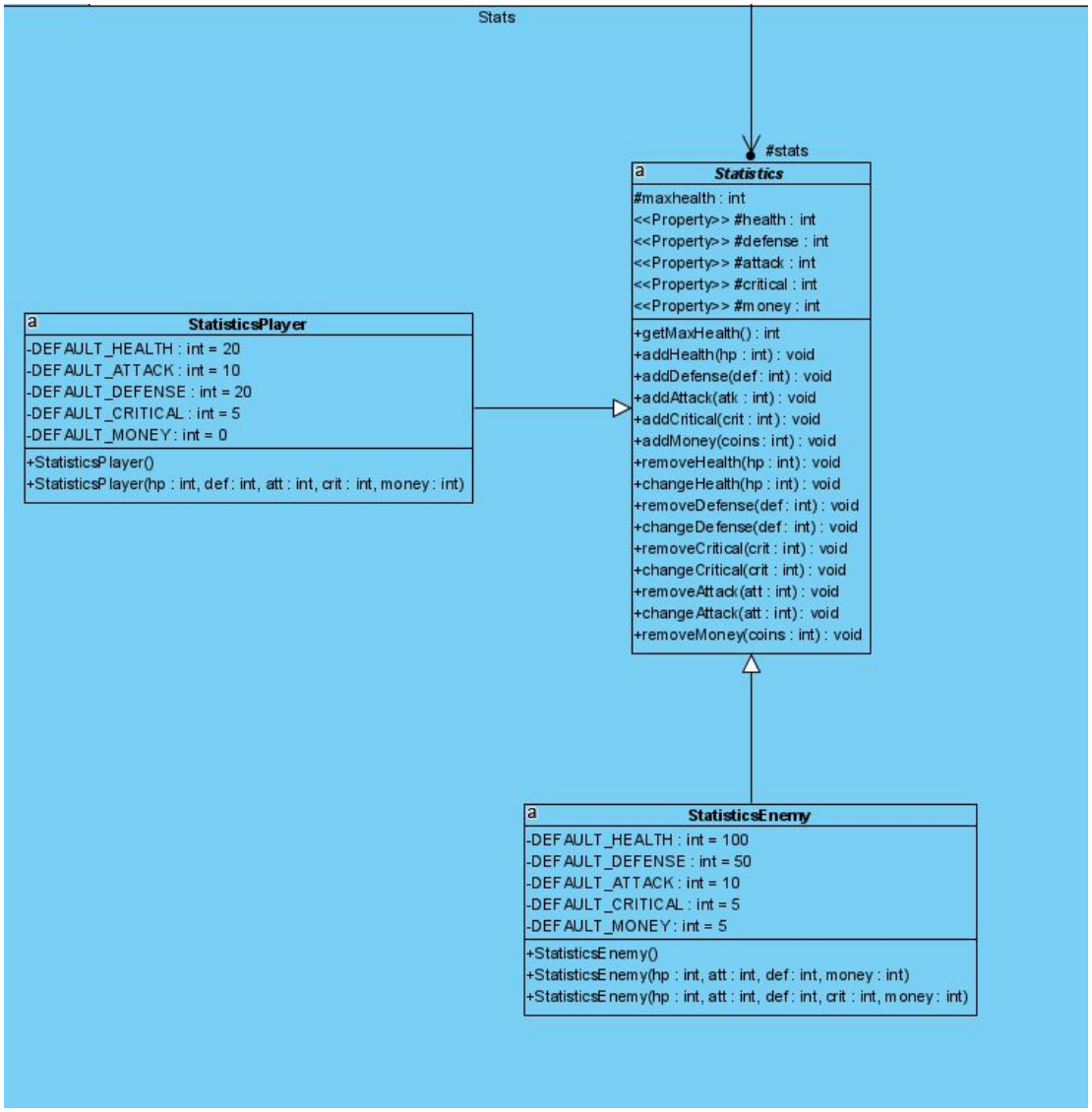


Package contenant les classes des lieux du jeu

Map: Classe représentant les "cases" composant le monde du jeu. Elle contient le nom du lieu, la description du lieu, trois HashMap contenant les ennemis présents, les objets au sol et les PNJ.
Elle a aussi quatre *Door* représentant les quatre points cardinaux, menant vers d'autres *Map.* A noter qu'une référence peut rester à *null* si on souhaite, par exemple, que le nord de la *Map* ne mène nulle part, ceci étant géré par la méthode actionGo() de *ActionManager.* Finalement elle contient aussi une référence vers un *Shop*, une boutique où le joueur pourra marchander. Là encore *Shop* peut rester à *null* si on ne souhaite pas qu'une boutique soit présente dans la *Map.*

Shop : Classe héritant de *Map*, celle-ci représente une boutique où le joueur peut négocier. Elle dispose d'une HashMap contenant les objets vendus par la boutique, une constante fixant le prix des potions vendues (on considère que chaque boutique en a un stock illimité) et une référence à une *Map* faisant office de sortie.

Stats :



Package contenant les classes statistiques du jeu

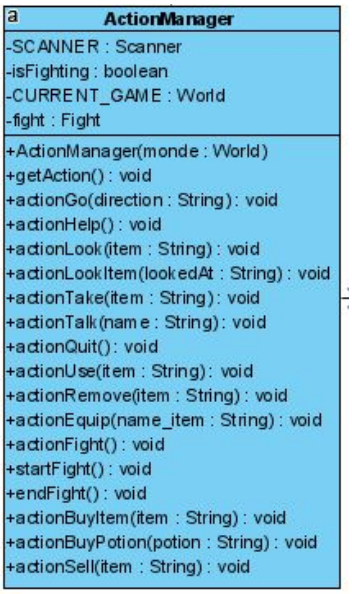
Statistics : Classe abstraite dont dérivent toutes les autres du package. Elle contient les statistiques dont disposent toutes les entités vivantes du jeu. Une constante contenant la vie maximale, des attributs contenant la vie, le score de défense, d'attaque, d'attaque critique et l'argent.

StatisticsPlayer : Classe héritant de *Statistics.* Elle contient cinq constantes définissant les statistiques par défaut du joueur, si celles-ci n'étaient pas spécifiées lors de son initialisation (constructeur par défaut).

StatisticsEnemy : Classe héritant de *Statistics.* Elle contient cinq constantes définissant les statistiques par défaut des ennemis, si celles-ci n'étaient pas spécifiées lors de leur initialisation (constructeur par défaut).

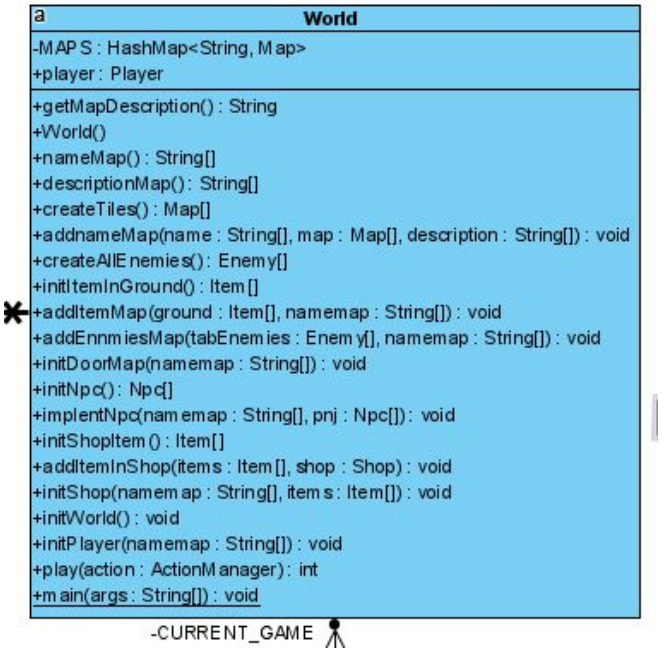
Classes diverses :

ActionManager : Classe centrale du jeu, elle contient un *Scanner*, un flag déterminant si le joueur combat ou non, une référence vers *Fight*, gérant les combats et une référence vers *World,* le monde du jeu. Dans getAction(), la classe récupère les lignes de commande entrées par le joueur avec le *Scanner*, les sépare dans un tableau où chaque mot est récupéré (les mots sont séparés par la commande String.split() de Java ainsi que le regex : "␣", deux mots sont donc identifiés par l'espace qui les sépare).
Avec le premier mot entré, on détermine l'action voulue par le joueur et on aiguille ainsi le second mot (ou le reste de la commande pour les actions composées) vers la méthode d'action correspondante. Il y a 11 actions possibles au total, toutes ne sont pas accessibles au même moment, selon qu'on se bat ou non.



Fight : Classe gérant les combats entre le personnage et les ennemis. Elle contient une hashmap d'ennemis (référence de la hashmap ENEMIES contenu dans *Map*) présent sur le lieu, une variable de dégâts infligés, un flag pour garder en mémoire si le joueur s'est défendu au "tour" précédent, 6 constantes et une référence constante vers le joueur.
Elle gère des actions d'attaque, de défense, la vérification s'il y a des ennemis à combattre (hashmap vide ou non), si le joueur ou l'adversaire est mort, la variation des statistiques de chaque combattant et l'affichage des statistiques à chaque "tour".

World : Classe représentant le monde du jeu. Elle dispose d'une hashmap contenant l'ensemble des *Map* du jeu ainsi qu'une référence *Player* vers le joueur.
Elle sert quasi exclusivement à initialiser le contenu du jeu. C'est donc ses méthodes d'initialisation qu'il faut utiliser pour ajouter ou retirer du contenu.



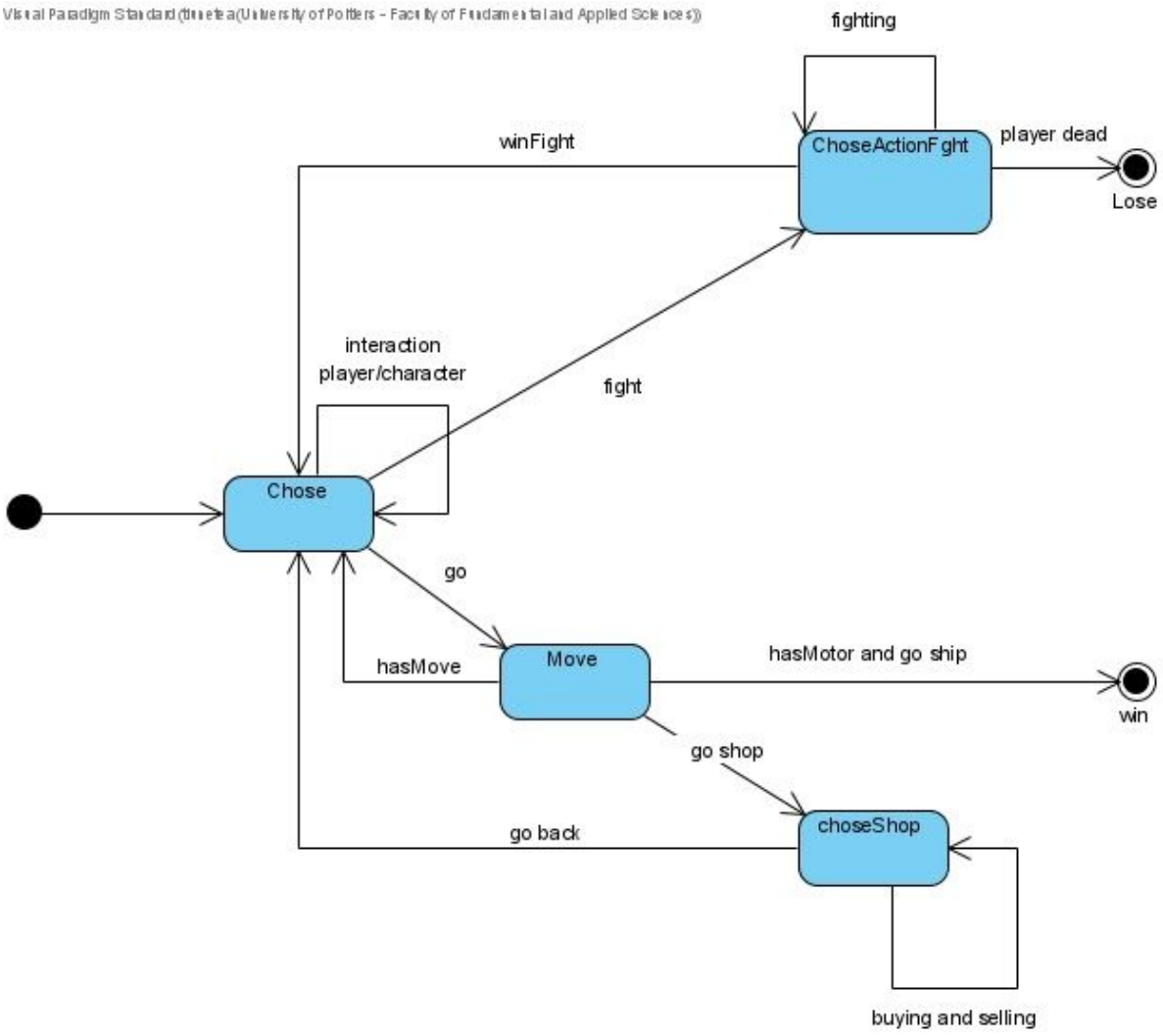


Diagramme d'état du joueur.

3. Organisation du groupe

Nous nous sommes organisés grâce à discord pour communiquer à distance et avec github pour pouvoir partager notre code.
Voici le lien de notre dépot : https://github.com/thbl088/pooProject1

Lors de ce projet, Thomas a converti le diagramme UML en java (en utilisant Visual Paradigm) et a travaillé sur les combats, une grande partie d'ActionManager, l'écriture des descriptions des zones de la carte, des dialogues et de plusieurs fonctions mineures ainsi qu'au débuggage. Il a aussi écrit l'introduction et le document utilisateur du rapport, le power point, le diagramme UML final et le diagramme d'état.

Pour commencer Charlie , s'est occupé principalement des packages suivant Characters , Items , Stats avec leur tests respective plus celui de FightTest . Ainsi il a pu faire l'initialisation du monde , cela a conduit à la gestion des maps et de leurs mise en forme en ajoutant leurs caractéristiques (nom, description , la connaissance de leur voisine , liste des pnj , ennemis , objets au sol) , initialisation / gestion de la boutique (items banque , dialogue , map voisine) . Implements des stats d'ennemis avec leurs objet respective , gestion du joueur (stats , modifier l'inventaire , équiper les armes , interaction avec la boutique). Aussi les pnj ajouts de dialogue , donation de leurs items , et gestion des dialogues . Ensuite , il a fait la méthode permettant de pouvoir jouer au jeu (le main) une boucle tant que le joueur est fini.
Pour conclure, il a organisé les différentes fin possibles pour notre héros qui mettront fin au jeu et participé au débuggage.

Ensuite, Alexis, qui a dessiné le diagramme UML initial, s'est occupé des package Locations et Doors. Ces packages contiennent les classes Map, représentant les cases de notre monde; Shop, des sous-map représentant des boutiques où le joueur peut échanger des objets, ainsi que Door et LockedDoor permettant de se déplacer entre les Map. Il a aussi participé à l'écriture d'une grosse partie du ActionManager, classe permettant de gérer les commandes du joueur, effectué diverses corrections syntaxiques au sein du code, et rédigé la documentation développeur.

CONCLUSION

Suite à ce projet nous avons réalisé toutes les fonctionnalités que l'on voulait implanté dans notre jeu. De plus, notre jeu est assez complet, il possède une dimension de jeu de rôle où il faut s'équiper d'item de plus en plus puissant afin d'affronter les boss. Nous pouvons ramasser des objets au sol, avoir une description d'un objet, communiquer avec des pnj, recevoir des objets suite à des conversations ou des combats. Il y a aussi un système d'achat d'objet et de revente. Les combats bien que basiques peuvent se relever compliquer si mal préparez comme on ne voit pas ni son attaque ni sa défense ce qui peut mener à des situations où on se tue sur la défense d'un ennemi ou que l'ennemi passe notre défense et nous tue. De plus, nous avons intégré de nombreux secrets au sein de notre monde.

Démonstration

Voici la vidéo de test (en non répertoriée sur youtube) : https://youtu.be/qFrFkH3VJdw
Si la vidéo ne marche pas elle est aussi dans le dossier Video de ce projet.