M1

PROJET JAVA

Wizard Guild Manager

DESRUMEAUX Thomas

BERTHEBAUD Yanis

Table des matières

Introduction

Notre projet est un jeu que nous avons imaginé, ce n'est pas un sujet proposé. Nous avons choisi un sujet qui nous parle tout en ayant en tête d'utiliser toutes les notions abordées en cours. Le principal objectif de ce projet est donc d'utiliser tout ce qu'on a vu en cours, ainsi que d'avoir un code le plus versatile possible.

Notre projet n'étant pas un sujet proposé, nous avons dû l'imaginer et le développer, pour cela nous avons commencé par faire une description générale du jeu, avec les principales fonctionnalités. Suite à cela, nous avons commencé des diagrammes avec les différentes classes qui entreraient en compte, pour déterminer les relations entre elles. Puis nous avons fini par déterminer quelles actions/méthodes chaque classe nécessiterait.

Nous allons commencer par une faire description générale du jeu, pour enchaîner sur une présentation du diagramme du projet, puis nous expliquerons certaines méthodes de classes et pour finir nous allons faire un point sur l'avancement du projet.

I. Description du jeu

Notre projet de java est un jeu de management d'une guilde de mage. Dans le jeu vous serez donc tout d'abord amené à créer votre guilde puis créer son maître. Vous pourrez ensuite, en prenant le contrôle du maître de la guilde, recruter un membre supplémentaire pour la guilde pour 50 pièces d'or. Ce membre peut être un aventurier (un mage) ou un employé.

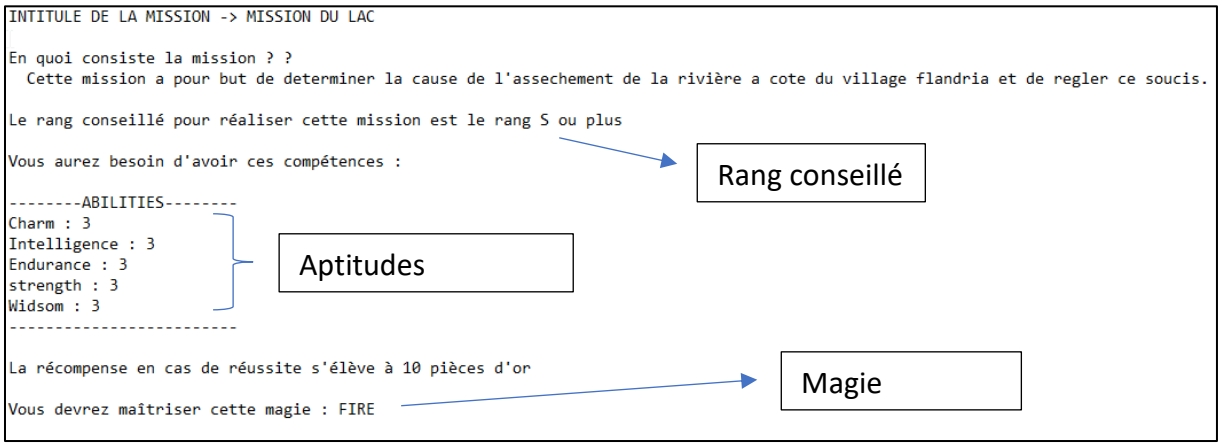
L'aventurier est un membre qui possède plusieurs caractéristiques, comme des niveau aptitudes (force, intelligence, etc.), un type de magie, un niveau (appelé Tier dans le jeu), etc. L'aventurier, quand il sera contrôlé par le joueur, pourra partir en mission et en fonction de son niveau et de la difficulté de la mission, il pourra la réussir ou la rater. En cas de réussite, il rapportera des pièces d'or à la guilde et il prendra de l'expérience pour monter de niveau. En cas d'échec, la guilde perdra des pièces d'or.

L'employé est un membre qui possède un métier (chanteur, barman, serveur, etc.), il possède en commun avec le maître de guilde la possibilité de de nous montrer la liste des missions disponibles, et si on le souhaite de nous écrire le détail d'une mission sur un manuscrit.

Une mission possède des caractéristiques comme un aventurier, ce qui définit la difficulté de celle-ci.

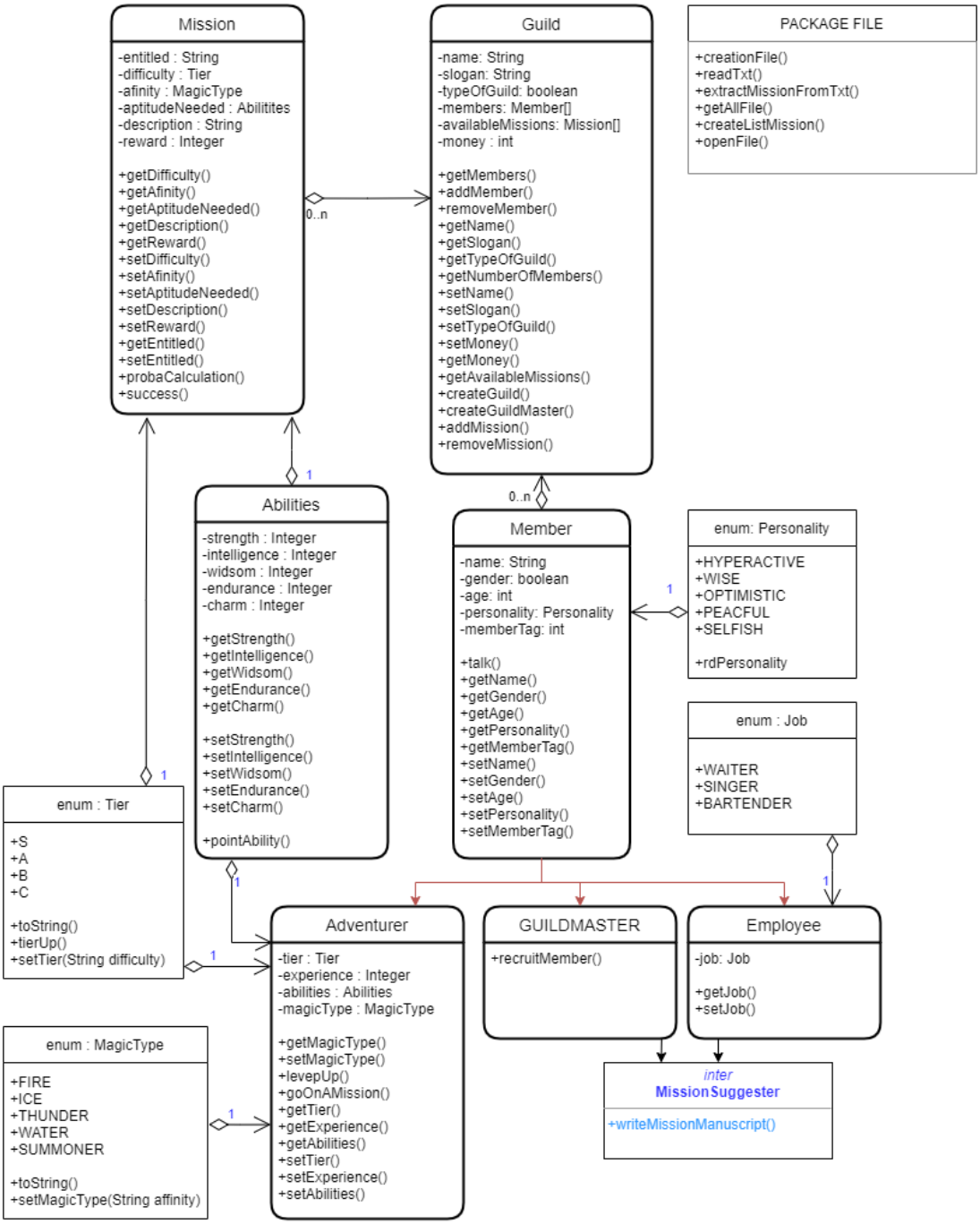
Pour résumer, le but est de faire partir ses aventuriers en mission pour qu'ils ramènent de l'argent à la guilde pour agrandir la guilde, et pour qu'ils montent de niveau, pour pouvoir effectuer des missions plus difficiles. Sachant qu'avoir différent mage est important pour avoir plus de type de magie, et ainsi pouvoir réaliser des missions qui nécessite un certain type de mission.

Exemple de manuscrit de mission :

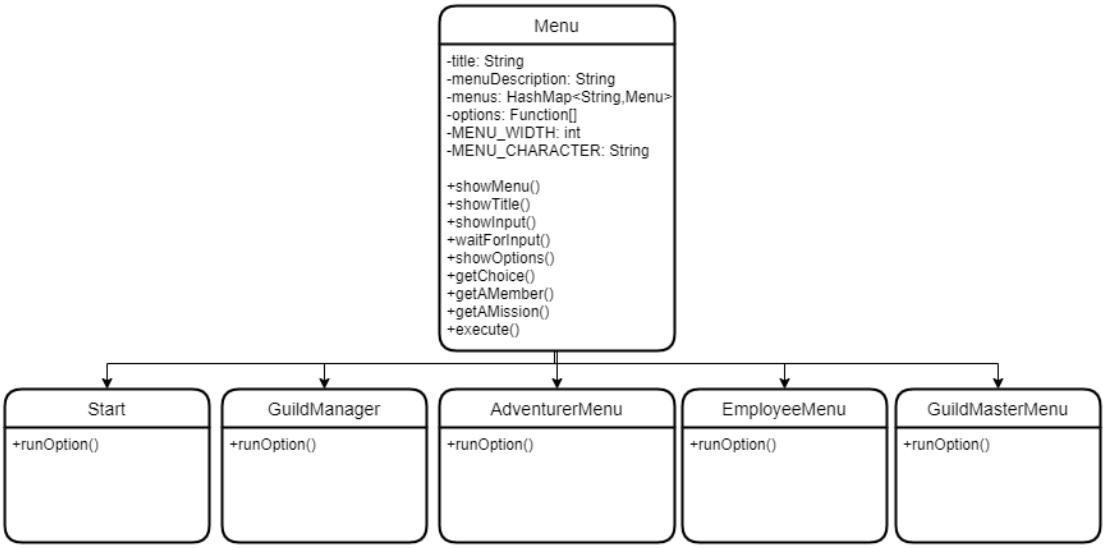


Si on ne respecte pas les caractéristiques conseillées, on a moins de chance de réussir la mission.

II. Diagramme du projet



Héritage


Menu
-title: String
-menuDescription: String
-menus: HashMap<String,Menu>
-options: Function[]
-MENU_WIDTH: int
-MENU_CHARACTER: String
+showMenu()
+showTitle()
+showInput()
+waitForInput()
+showOptions()
+getChoice()
+getAMember()
+getAMission()
+execute()
Start
+runOption()
GuildManager
+runOption()
AdventurerMenu
+runOption()
EmployeeMenu
+runOption()
GuildMasterMenu
+runOption()

III. Explications des classes

Class Mission :

En plus des accessors et mutators, la class Mission possède les méthodes probaCalculation() et success(), qui permettent de donner, pour une mission donnée et un aventurier donné, la probabilité de réussite et si la mission est réussie ou non.

Class Abilities :

Cette class possède une méthode, pointAbility(), qui est la méthode qui permet, lors de la création d'un aventurier, de renvoyer un objet de type Abilities avec les différentes valeurs entrées par l'utilisateur.

Class Tier :

La première méthode est tierUp(), elle permet de renvoyer le niveau supérieur d'un aventurier donné. La seconde méthode est setTier(), qui permet de donner le Tier correspondant au String en argument.

Class MagicType :

La méthode setMagicType() permet de passer d'un String à un MagicType.

Package File :

Dans ce package on retrouve la class Fichier avec 6 méthodes :

- creationFile() > création d'un fichier txt dans le répertoire Display
- readTxt() > met les informations du fichier txt dans une liste
- extractMissionFromTxt() > prend la liste avec les infos et les utilise pour créer une mission
- getAllFile() > récupère le nom des fichiers dans le répertoire Mission et les met dans une liste
- createListMission() > création d'une liste avec toutes les missions extraites du répertoire
- openFile() > ouvre sur le bureau le txt en argument

Class Personality :

La méthode rdPersonality() renvoie une personnalité aléatoire.

Interface MissionSuggester :

La méthode writeMissionManuscript() permet d'écrire « proprement » dans un txt les informations d'une mission, et ouvre le txt correspondant.

Class Member :

En plus des accessors et mutators, la classe Member possède une fonction talk() qui permet à un membre de se présenter. Cette fonction est écrasée par les classes héritantes de Member pour fournir des dialogues personnalisés pour chaque type de membre.

Class GuildMaster:

Cette classe hérite de la classe Member. Un Maître de guilde peut, en plus des autres membres, recruter un aventurier ou un employé dans la guilde en échange de pièces d'or (50) grâce à la méthode recruitMember() . De plus, il peut également imprimer une mission que la guilde possède via la méthode writeMissionManuscript() de l'interface MissionSuggester.

Class Employee:

Cette classe hérite de la classe Member. Un employé possède un travail (barman, serveur ou chanteur), il peut également imprimer une mission que la guilde possède via la méthode writeMissionManuscript() de l'interface MissionSuggester.

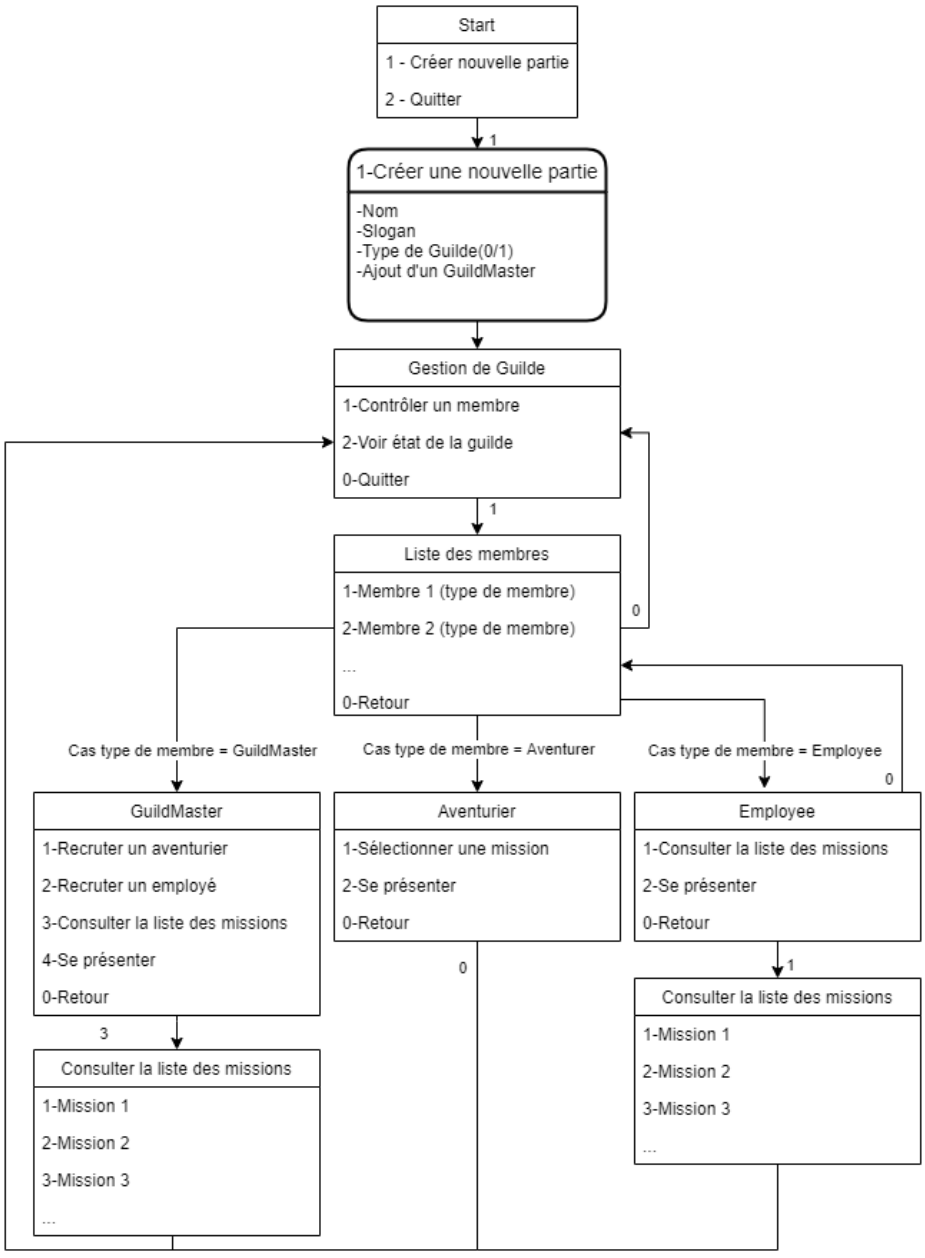
Class Adventurer :

Cette classe hérite de la classe Member. La méthode levelUp() permet de monter d'un niveau l'aventurier s'il a assez d'expérience.

Class Menu

Pour pouvoir réaliser un menu versatile, factorisé et optimisé, nous avons fait le choix de réaliser une class Menu permettant d'afficher un menu via la méthode showMenu() puis de récupérer le choix de l'utilisateur par getChoice().Enfin, nous exécutons la fonction correspondante qui se trouve dans runOption(), une méthode spécifique à chaque menu.

Le menu de notre jeu et les liens entre menus se trouvent dans le diagramme suivant :



IV. Avancement du projet

Nous avons pu réaliser le programme « minimum » pour que le jeu soit jouable. Nous avons utilisé le maximum de notions vues en cours. Nous aurions aimé ajouter plus de possibilités pour la jouabilité, comme le fait qu'un aventurier puisse devenir amis avec un autre aventurier et ainsi pouvoir faire des missions en coopérations. Nous voulions également, derrière le choix du type de guilde, officielle ou clandestine, modifier « l'ambiance » du jeu, notamment avec des dialogues différents, et des intitulés de missions différents. Néanmoins notre programme est assez versatile, notamment au niveau du menu, ce qui permet d'ajouter facilement plus d'étapes à notre programme, et aussi les missions qui sont facilement ajoutables, car il suffit d'ajouter un fichier .txt avec les mêmes conventions que ceux existant dans le fichier mission et le programme les lira.

Conclusion

Nous sommes satisfaits de notre projet, nous avons réussi à nous organiser pour travailler tous les deux le plus efficacement de la même manière sur le programme. Pour cela nous avons utilisé des outils de collaboration comme GitHub (Lien du projet) pour pouvoir avancer sur deux branches séparées et les regrouper à la fin pour obtenir un programme complet. Ce projet nous a également permis de comprendre plus en profondeur le langage Java et les notions du cours.