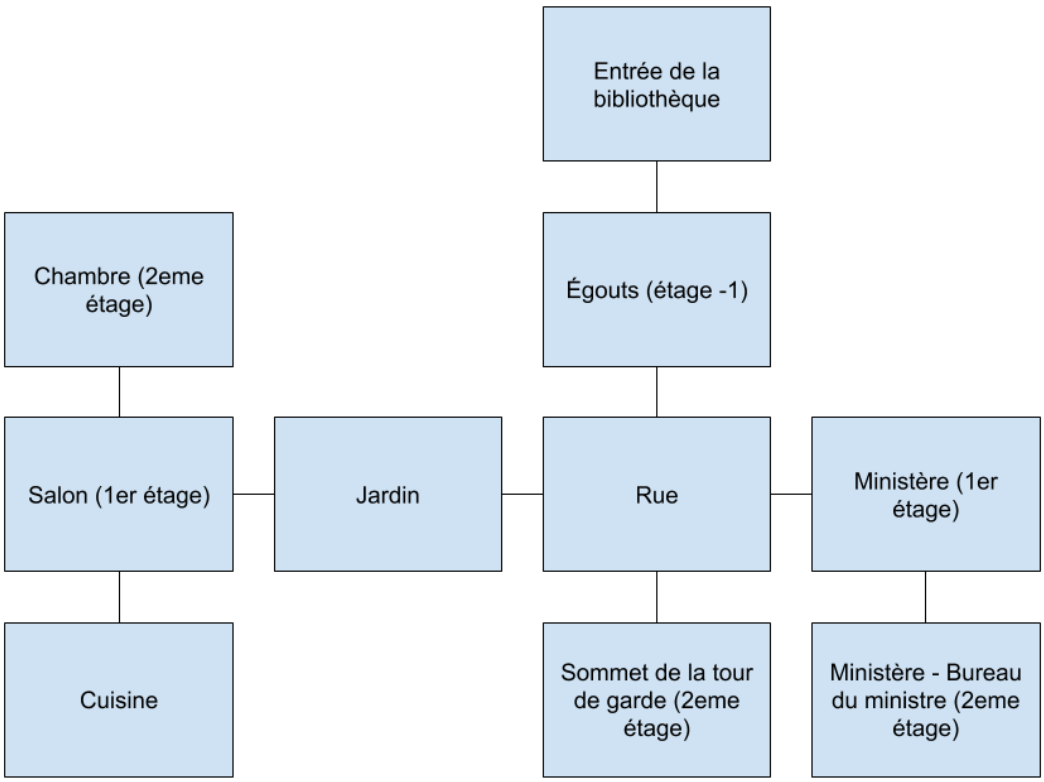
Antisophia
(A3P(AL) 2018/2019 G2A)

Accès javadoc : https://perso.esiee.fr/~desplanl/doc/

Table des matières

I - Informations

A) Auteur : Louis DESPLANCHE
B) Phrase thème : Transmettre le savoir aux citoyens
C) Résumé du scenario : Une jeune femme vivant dans un état totalitaire, où les citoyens sont privés de connaissances, va découvrir la culture et la diffuser.
D) Voici le plan :



E) Dans un état sans culture, asservissant ses citoyens grâce à la manipulation et la désinformation. Une jeune femme tombe sur un article caché sous le planché en faisant du rangement. Cet article vieux de 200 ans, relate de la vie en démocratie et parle de nombreux livres (si le joueur ne parvient pas à trouver alors la partie s'arrête). L'héroïne, intéressée par cette culture, décida de partir à l'aventure pour en découvrir plus. Pour chercher des informations celle-ci part se renseigner au ministère qui lui confisque le papier sans lui donner davantage de détail. En rentrant chez elle, vient la voire un homme mystérieux qui lui parla du papier et lui dit de se rendre le lendemain dans les égouts (si le joueur ne va pas à l'endroit indiquer le lendemain elle perd).
Le lendemain dans les égouts l'homme lui parla du morceau de journal qu'il avait trouvé lui-même quelques années plus tôt. Depuis ce jour il enquêtait et entendu parler d'une bibliothèque cachée avec tous les livres de l'ancien régime.
Ils se mirent en tête de trouver ce mystérieux bâtiment. Pour cela, le lendemain, l'héroïne prit une brique dans son jardin pour assommer un garde dans la tour puis lui voler sa tenue afin de rentrer dans le ministère. Elle rentre dans le bureau du ministre pour voler les archives. Puis rentre chez elle pour les lires. Et se rend sur le lieu de la bibliothèque révélé

par ce qu'elle a trouvée. Elle prend les livres, les porte jusqu'à la tour puis les répand pour sortir la population de l'ignorance. La police politique ne sachant pas comment réagir tire sur les passants qui tentent de récupérer les ouvrages au sol. Ceci créer une foule en colère qui parvient à prendre contrôle du ministère et renverser le gouvernement.

F)

Lieux :

- Chambre : où le joueur peut passer les jours
- Salon : lieu de départ de l'aventure
- Cuisine : lieu pour manger
- Jardin : endroit de passage
- Rue : élément central de la rébellion
- Egouts : lieu de rencontre et de passage
- Bibliothèque
- Ministère : hall du bâtiment
- Bureau du ministre : Salle des archives

Items :

- Article de journal
- Nourriture déshydratée
- Tenue de policier
- Parpaing
- Clé de la bibliothèque
- Archives

Personnages :

- Héroïne
- Etrange personne
- Peuple
- Ministre
- Réceptionniste

II – Exercices

7.5

Avant printInfoLocation() il existait une duplication de code entre printWelcome() et goRoom(). Cette fonction a donc vocation à être appelé par celles-ci

7.6

On créé getExit(String pDescription) pour ne plus avoir besoin d'accéder aux attributs des salles environnantes (cause de potentiels bug). Pour cela la fonction associe à chaque point cardinal sous forme de String une salle.

7.7

Room s'occupe de gérer les salles, il est donc logique de lui demander de lui demander des informations à propos de celles-ci. Game quant à lui s'occupe de gérer les actions de l'utilisateur ainsi que les sorties. Il est donc plus logique que Game affiche les informations.

7.8

On remplace les attributs des salles à proximité par un HashMap<String, Room> ce qui permet plus de flexibilité dans le nommage des directions et dans leurs stockages.

7.8.1

On ajoute les nouvelles directions haut et bas dans le constructeur puis on modifie la fonction goRoom()

7.9

KeySet est comme un tableau mais qui peut être redimensionné à volonté. Un keySet<object> peut s'apparenté à un HasMap<int, object>

7.10

La fonction vient lister les clés de la HashMap dans une liste Set<String>. Chaque élément vient ensuite être ajouté à la String.

7.10.2

La javadoc permet à tous les développeurs de comprendre et de pouvoir utiliser les classes créées.

7.11

Ajout de getLongDescription() ce qui nous évite la duplication de code.

7.14

Ajout de la fonction look() ceci nous permet d'afficher toutes les information de la salle explorée. Ceci utilise juste getLongDescription(). On modifie également la liste des fonctions disponibles.

7.15

Ajout de la fonction eat() qui affiche un message. On modifie également la liste des fonctions disponibles.

7.16

Ajout de de la fonction showAll() dans CommandWords qui affiche la liste des fonction disponibles. Mais CommandWords n'est accessibles que depuis Parser alors on ajoute showCommands() dans cette classe pour pouvoir être appelé depuis Game.

7.17

7.18

La fonction getExitString() liste les commandes contenues dans sValidCommands sous forme de chaine de caractères.

7.18.2

Le StringBuilder est comme une chaine avec un nombre de caractère prédéfini. Il est donc plus facile pour l'ordinateur de les modifier et de les additionner. Quand associé deux chaines de caractère revient à en créer une troisième contenant les deux autres, le StringBuilder permet de faire ces opérations sans créer de nouvelles variables. Le principal défaut réside dans son manque de flexibilité.

7.18.5

On déplace createRooms() dans la classe Room. Pour que les salles restent malgré tout accessible dans le reste du programme, on crée une variable statique de type HashMap<String, Room>

contenant les salles associées à leurs noms. Il faudra faire attention dans la suite du programme de ne pas appeler le HashMap avant d'avoir appelé createRooms()

7.18.7

addActionListener() associe à un objet graphique une classe hérité de ActionListener, à chaque évènement la methode actionPerformed(final ActionEvent pE) de l'objet est exécutée.

7.18.8

J'ai ajouté un bouton de « Aide » placé dans un JLayout dans lequel j'ai aussi placé l'EntryField. Ceux-ci sont placé en bas de la fenêtre. J'ai associé un MouseListener au bouton, ainsi celui-ci exécute la commande « aide » comme si celle-ci était tapé dans l'EntryField

7.19

L'architecture permettrait au projet d'avoir une plus grande facilité à évoluer. De plus elle serait plus facile à traduire

On pourrait créer une classe contenant tous les messages à afficher.

7.19.1

Le changement d'architecture demande de repenser une bonne partie du code. L'architecture MVC demande trois classes :

- La vue qui va gérer l'interface graphique et donc ce que voit l'utilisateur (UserInterface)
- Le modèle qui possède les informations à afficher (GameModel)
- Le contrôleur qui va gérer traiter les actions de l'utilisateur (GameEngine)

7.20

Ajout de la classe Item, avec deux attributs : le poid et la description. Mais également des accesseurs et un constructeur naturel

7.21

Les items doivent être construits lors de l'initialisation et doivent êtres affichés par GameModel

7.22

Ajout d'une liste afin de stocker plusieurs items au sein des salles

7.23

Pour ajouter une commande de retour j'ai créé un nouvel attribut dans GameModel chargé de stocker la salle précédemment visitée. À chaque fois que le joueur va changer de salle, la salle actuelle va être stocké dans cette variable avant toute action.

Quand l'utilisateur demande le retour, il va aller dans la salle stockée dans la variable.

7.24

Quand on rajoute des caractères après la commande retour, la commande fonctionne comme ci de rien n'était

7.25

La fonction ne permet pas de revenir très loin en arrière : quand on avance de trois salles et que l'on essaie de faire deux retours on se retrouve à la troisième salle

7.26

Le stack permet de créer des piles de données. On ajoute à chaque changement de salle, l'ancienne salle dans la pile. Lorsqu'on veut revenir dans la salle précédente alors on place le joueur dans la pièce au sommet de la pile, puis on supprime l'élément.

7.27

L'automatisation des tests permet de vérifier qu'il n'existe aucune régression à chaque mise-à-jour du code sans que le développeur ne soit obligé d'y passer beaucoup de temps.

7.28

Pour automatiser les tests, on pourrait modifier la classe UserInterface afin de router les données de sorties vers un classe Test. Les sorties seraient comparées aux sorties attendues stockées dans des fichiers texte.

7.28.1

Pour mettre en place le système de test j'ai écrit une nouvelle commande en modifiant "GameEngine" et "Command".

Dans GameEngine, j'ai rajouté une fonction permettant de lancer les tests : le système commence à vérifier si le fichier renseigné dans la commande existe, sinon il teste le nom du fichier suivit de ".txt". Si le fichier n'est toujours pas trouvé alors on affiche un message d'erreur à l'utilisateur. J'ai choisi de ne pas inclure cette partie dans un "try/catch" pour permettre une plus grande flexibilité dans mon programme.

Une fois que l'on a validé le fichier le programme va lire ligne à ligne le contenu puis l'exécuter.

J'ai aussi rajouté un système permettant de lire un fichier au format XML (vous trouverez les spécifications du XML dans le code. Dans un souci d'intéropérabilité avec ce qui est demandé dans l'énoncé ceci n'est pas activé par défaut et les fichier txt sont prioritaires dans la conception même du code. L'avantage du XML est de pouvoir rajouter à coté, si l'on le souhaite, le retour attendu et de le comparer avec le retour obtenu.

7.29

Afin de simplifier GameModel on ajoute une classe Player qui stocke les variables relatives au joueur. Il ouvre même la possibilité de pouvoir ajouter plusieurs personages jouable à l'aventure.

La classe contient la pièce actuelle, la liste des salles précédentes et un nom

Pour se déplacer, on copie juste les fonctions de la classe GameModel (goBack(), changeCurrentRoom(...), etc)

7.30

Pour ajouter la possibilité au joueur de porter un objet, on ajoute une variable de type item à la classe Player. Il est aussi nécessaire de rajouter deux fonctions permettant au joueur de ramasser ledit objet et de le jeter. Ces fonctions se contenteront de transférer un objet de type Item de la pièce vers le joueur ou inversement.

7.31

Afin de porter plusieurs items il faut, comme pour la classe Room transformer l'attribut Item en HashSet<Item>. Cela permet de créer un tableau d'item. Il est également nécessaire de changer les accesseurs.

7.31.1

Pour mettre en commun les fonctions de recherche d'items dans le tableau, il est nécessaire de créer une classe "ItemList". Il s'agit d'une classe contenant un tableau d'item, des accesseurs et des algorithmes de recherche (tel la fonction pour chercher un item grâce à son nom)

7.32

Pour ajouter un poids maximal dans le jeu, il faut commencer par rajouter un attribut le contenant dans la classe Player (ainsi que ses accesseurs). On ajoute également une fonction dans ItemList permettant de calculer le poid total de la liste. Ceci nous permettra de voir combien porte déjà le joueur. La fonction take doit être modifié afin de tester si, lorsque le joueur tente de ramasser un item, si la somme de celui-ci et ceux déjà présent dans l'inventaire n'excède pas le poids maximal.

7.33

Pour ajouter un inventaire, comme toutes les autres commandes, il faut le rajouter dans la classe CommandWord et modifier processCommand() dans GameEngine. La fonction pour lister les éléments de l'ItemList a déjà été implémenté précédemment.