La grève

Projet d'IPO(A3P)

François Soulié
perso.esiee.fr/ soulief/IPO/francois

2019/2020

Tables des matières

1. Mon projet

1.1 Thème

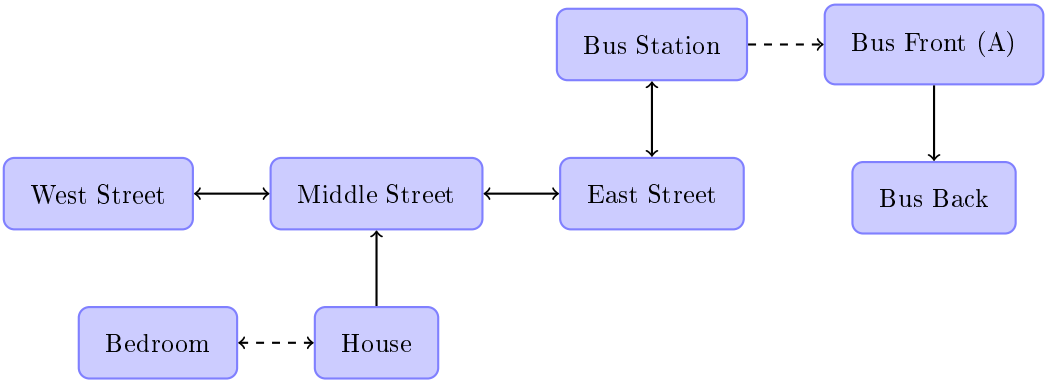
"C'est la grève des bus et vous devez arriver à l'heure à l'ESIEE."

1.2 Résumé du scénario

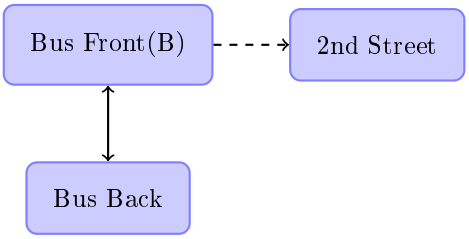
Il est 7h15 et vous venez de vous réveiller : vous êtes en retard pour prendre le bus jusqu'à l'ESIEE. De plus, c'est le premier jour de la grève des bus, vous ne pouvez pas rater celui de 7h34 puisque c'est le seul de la journée. Vous aurez l'occasion de rencontrer plusieurs personnes sur le chemin, comme par exemple un contrôleur où même des manifestants bloquant la route.

1.3 Plan

1.3.1 1ère partie



1.3.2 2ème partie



1.3.3 Dernière partie

1.4 Détails des lieux, items, personnages

1.4.1 Lieux

- « Street - West » : la partie Ouest de la rue.
- « Street - Middle» : la partie centrale de la rue.
- « Street - East» : la partie Est de la rue.
- « Bus Station » : la gare routière.
- « Bus » : l'intèrieur du bus.
- « Bus - Front » : l'avant du bus.
- « Bus - Back » : l'arrière du bus.
- « 2nd Street » : la deuxième rue, terminus du bus
- « 3rd Street » : la troisième rue, menant à l'ESIEE
- « ESIEE Paris » : l'ESIEE

1.4.2 Personnages

- Controleur
- Manifestants

1.4.3 Items

- Pass Navigo
- Clés
- Sac de cours
- Beamer
- Cookie

1.5 Situations gagnantes et perdantes

1.5.1 Pour gagner

Arriver à l'heure à l'ESIEE

1.5.2 Pour perdre

- Oubli du pass navigo (Impossible de prendre le bus)
- Oubli du sac de cours

1.6 Enigmes, mini-jeux, combats

Des choix moraux et logiques: penser à prendre son sac de cours, ne pas frauder, ...

2. Rapport d'exercices

Exercice 7.5

Création de la procédure printLocationInfo() dans Game: affiche la description de la salle actuelle ainsi que ses sorties disponibles.
On l'utilise ensuite dans les fonctions goRoom() et printwelcome() afin d'éviter la répétition de code.

Exercice 7.6

Création de la méthode getExit() dans Room: retourne la salle se trouvant dans la direction entrée en paramètre, ou null si il n'y a pas de salle / si cette direction n'existe pas.
Cette méthode permet de mettre les attributs d'une salle en privé.

Exercice 7.7

Création de la méthode getExitString() dans Room: retourne une chaine de caractères contenant toutes les sorties disponibles de la salle sur laquelle elle a été appelée.
Cette méthode nous permet d'éviter de répéter du code dans printWelcome() et printLocationInfo().

Exercice 7.8

Modification des attributs de Room: toutes les sorties sont maintenant stockées dans une HashMap<String,Room> ce qui simplifie les getters et setters.

Exercice 7.9

Modification de getExtiString() dans Room: utilisation de la structure Set pour simplifier la méthode.

Exercice 7.10

La méthode getExitString() retourne une chaîne de caractères contenant toutes les sorties disponibles de la salle sur laquelle elle a été appelée.
Pour cela, elle utilise la fonction keySet() sur la HashMap contenant les sorties de cette salle, ce qui retourne un objet similaire à un tableau contenant chaque

direction définies dans Game.
Puisque la structure d'un Set est similaire à celle d'un tableau, on peut utiliser une boucle foreach afin de former la chaîne de caractères qui sera retournée.

Exercice 7.11

Création de la méthode getLongDescription() dans Room: permet de simplifier la méthode printLocalInfo() de Game.

Exercice 7.14

Création de la commande look: permet au joueur de savoir où il se trouve.

Exercice 7.15

Création de la commande eat: permet au joueur de manger.

Exercice 7.16

Création de la méthode showAll() dans CommandWords: affiche tous les premier mots des commandes.
Création de la procédure showCommands() dans Parser: permet d'afficher toutes les commandes du jeu dans la procédure help() de Game.

Exercice 7.18

Modification de la fonction showAll() de CommandWords: nous l'avons renommée getCommandList() et elle retourne maintenant un String contenant toutes les commandes du jeu, séparées par un espace. Cette commande est en quelque sortes la getExitString() de CommandWords.

Exercice 7.18.1

Création d'une HashMap de Rooms dans Game contenant toutes les salles du jeu associées à leur nom. Les méthodes de Game peuvent maintenant accèder à ces dernières.

Exercice 7.18.6

Intégration de zuul-with-images dans le projet. Les modifications principales sont:

- Le jeu se joue maintenant sur une interface graphique et plus dans l'invite de commande.
- La classe Game ne sert plus qu'à lancer le jeu; GameEngine rempli son ancienne fonction.
- processCommand() a été renommée interpretCommand(), son paramètre est maintenant un String. On utilise alors la fonction getCommand() de Parser pour obtenir la commande correspondant à ce String.
- On n'affiche plus à la console mais à l'interface graphique.
- Le Parser lit maintenant le champs de text de l'interface graphique.

Exercice 7.18.7

La fonction AddActionListener() indique au programme d'appeler la procédure actionPerformed() de son paramètre lorsque qu'une action est effectuée sur l'objet ciblé.
On a comme paramètre this car la classe UserInterface implémente ActionListener et possède ainsi la fonction actionPerformed().

Exercice 7.18.8

J'ai ajouté 3 boutons: Inventory, Help et Exit qui correspondent à leurs commandes respectives. Pour les placer dans une seule colonne, j'ai créé un deuxième JPanel les contenant, que j'ai ensuite associé à l'est de vPanel.
J'ai aussi tenté d'ajouter des boutons de déplacement pour au final me rendre compte que ces derniers allait contre le principe du jeu.

Exercice 7.20

Création de la classe Item ayant comme attribut un poids
Ajout d'un attribut de type Item dans Room, accompagné de son getter et de son setter.

Exercice 7.21

Ajout d'un attribut Description dans Item.

Exercice 7.22

Modification de l'attribut de type Item dans Room: c'est maintenant une HashMap<String,Item>. De cette façon, une salle peut contenir plusieurs items. Le setter et le getter ont naturellement eux aussi changé. On a maintenant:

- addItem(String, Item) qui ajoute un Item à l'inventaire d'une salle.
- getItem(String) qui retourne l'item demandé.
- removeItem(String) qui supprime un Item de l'inventaire d'une salle.

Exercice 7.22.1

J'ai utilisé une HashMap dans l'exercice précédent car les méthodes put(), get() et remove() correspondent aux methodes demandées et rendent l'accès à un Item facile.

Exercice 7.23

Ajout d'une commande back au jeu: cette dernière permet au joueur de retourner dans la salle précédente.

Exercice 7.24

La commande back ne fonctionne pas corrèctement.

- Si on l'utilise dans la première salle, on obtient une erreur.
- Si on écrit un second mot après back, elle fonctionne quand même.

Exercice 7.25

Si on avance de 3 salles pour ensuite utiliser back 2 fois, on s'attends à retourner dans la première salle. Cependant, le premier back nous fait retourner dans la seconde salle, ce qui est normal, mais le deuxième back lui nous fait retourner dans la troisième salle.
Ce n'est pas satisfaisant, on souhaiterai pouvoir retourner à la première salle en utilisant la commande back.

Exercice 7.26

Modification de la commande back: on a ajouté à la classe GameEngine un attribut previousRooms de type Stack<Room> qui contient le chemin effectué. À chaque utilisation de back, on retourne dans la dernière salle enregistrée par previousRooms tout en la supprimant du Stack.
La commande back fonctionne maintenant comme attendu; on peut retourner à la première salle en l'utilisant.

Exercice 7.28

Création d'une commande test lisant dans l'ordre toutes les commandes indiquées dans un fichier .txt, afin de simplifier le test du jeu.
On met alors en place 3 fichiers de test:

- court.txt, permet de tester les commandes principales du jeu.
- ideal.txt, effectue le chemin idéal pour gagner le jeu.
- possibilites.txt, explore toutes les possibiliés du jeu.

Exercice 7.29

Création d'une classe Player: c'est maintenant cette classe qui contient l'attribut currentRoom et qui effectue les actions qui affectent cet attribut, c'est-à-dire les commandes go et back. Cependant, c'est toujours GameEngine qui s'occupe de fournir leurs paramètres et d'afficher le résultat de ces actions.

Exercice 7.30

Création de 2 nouvelles commandes take et drop qui permettent au joueur de ramasser un objets ou de déposer celui qu'il porte.
Ce changement nécéssite l'ajout d'un attribut item dans la classe Player de type Item et du getter et du setter lui correspondant. On à aussi évidemment ajouté les méthodes takeItem() et dropItem(), qui retournent le résultat de l'action, que GameEngine utilise.

Exercice 7.31

Modification de l'attribut de type Item dans Player: c'est maintenant une HashMap<String,Item>. De cette façon, un joueur peut maintenant porter plusieurs items. Le setter et le getter ont naturellement eux aussi changé. On a maintenant:

- addItem(String, Item) qui ajoute un Item à l'inventaire d'une salle.
- getItem(String) qui retourne l'item demandé.
- removeItem(String) qui supprime un Item de l'inventaire d'une salle.

Exercice 7.31.1

On remarque que les attributs items de Room et inventory de Player correspondent tous les deux à une HashMap<String,Item> et ont par conséquent les mêmes getters et setters, ce qui nous fait répéter du code.
Pour éviter cela, on crée une classe ItemList ayant pour seul attribut une HashMap<String,Item> et possèdant les méthodes que Room et Player ont en commun, c'est-à-dire les setters et getters de leur HashMap d'items.

Exercice 7.32

Ajout des attributs maxWeight et currentWeight dans Player, limitant ainsi la quantité d'objets que le joueurs peut porter.

Exercice 7.33

Ajout d'une méthode getItemString() dans ItemList qui retournera un String contenant tous les items de sa HashMap sous la forme "Item(poids)".

Exercice 7.35

Intégration de zuul-with-enums dans le projet. Enum nous permet de déclarer des constantes, ce qui est parfait pour les mots de commande.
Les modifications principales sont:

- Création de l'enum CommandWord contenant tous les mots de commande.
- Modification de validCommands dans CommandWords: c'est maintenant une HashMap<String,CommandWord>.
- Modification de Parser: plus besoin de if/else pour la Command retournée.

Exercice 7.35.1

Modification de interpretCommand() dans GameEngine: on utilise maintenant un switch pour interpreter les instructions du joueur.

Exercice 7.40

Modification de l'enum CommandWord: les constantes sont maintenant associées à leur valeur de type String.
Modification du constructeur de CommandWords: l'attribut validCommands est maintenant initialisé avec une boucle foreach lui ajoutant chaque couple [CommandWord.toString(), CommandWord].
Ce changement nous permet de n'avoir qu'à modifier l'enum CommandWord lorsqu'on veut modifier une commande ou en ajouter une.

Exercice 7.42

Ajout d'un attribut timeLimit dans GameEngine et d'un attribut currentTime dans Player. À l'avenir, lorsque currentTime dépasse timeLimit le game over sera déclaré.

Exercice 7.43

Ajout d'une méthode isExit() dans Room qui retourne vrai si la salle entrée en paramètre est une sortie de la salle sur laquelle la méthode à été appelée.
Modification de la commande walkBack() dans Player: elle utilise maintenant la nouvelle méthode isExit() pour vérifier que la salle précédente est une sortie de la salle actuelle.

Exercice 7.44

Création de la classe Beamer qui hérite d'Item. Cette dernière correspond à un objet capable de téléporter le joueur à une salle précédemment enregistrée. Ajout des commandes fire et charge au jeu, qui permettent comme leurs noms l'indiquent de charger puis d'utiliser le beamer.

Exercice 7.45

Création de la classe Door, elle correspond à une porte et peut être fermée.
Modification de setExit() dans Room: avec chaque direction de sortie est initilisée une porte lui correspondant.
Ajout d'un attribut de type HashMap<String,Door> dans Room.

Ajout d'une méthode lockDoor() dans Room qui permet de fermer la porte dans la direction indiquée en paramètre et de lui associer un objet qui pourra l'ouvrir.

Exercice 7.46

Création de la classe RoomRandomizer qui est initialisée à partir d'une HashMap<String,Room> et possède la méthode getRandomRoom() retournant une salle de la HashMap au hasard à chaque appel.
Création de la classe TransporterRoom héritant de Room modifiant la méthode getExit() qui ici retourne une salle du jeu au hasard grace à la classe RoomRandomizer.

Exercice 7.46.1

Création de la commande alea permettant d'annuler le hasard des TransporterRoom pour tester le jeu.

Exercice 7.47

Abstraction de la classe Command.
Création de la méthode abstraite execute() dans Command. Création de classe [Nomdelacommande]Command pour chaque commande du jeu, modifiant à chaque fois la méthode execute() par celle qui lui correspond dans GameEngine.
Suppression de toutes les méthodes de commandes dans GameEngine.

Exercice 7.47.1

Division du projet en plusieurs paquets: pkg_gameobjects, pkg_commands, pkg_interface, etc.

Exercice 7.48

Création de la classe GameCharacter correspondant à un personnage.
Ajout de la commande talk au jeu, qui permet au joueur de lire la ligne de dialogue associée à un personnage.
Ajout d'un attribut du type HashMap<String, GameCharacter> dans Room.
Modification de la méthode getLongDescription() dans Room qui affiche maintenant les personnages présent dans une salle.

Exercice 7.53

Création de la classe Main et de sa méthode statique initialisant une instance de Game.

3. Mode d'emploi

Les instructions relatives à l'installations et au lancement du jeu sont disponibles sur l'accueil du **site internet** de ce dernier.

4. Déclaration anti-plagiat

Ce sont des recherches sur la Javadoc officielle et sur des sites internets tels que **GeeksForGeeks** ou encore **StackOverflow** appuyant ces recherches qui m'ont permi de réaliser ce projet. Aucune partie de code trouvée sur internet n'a été réutilisée, hormis celle de zuul-bad evidemment.