RAPPORT - IPO 2018-2019 G1A

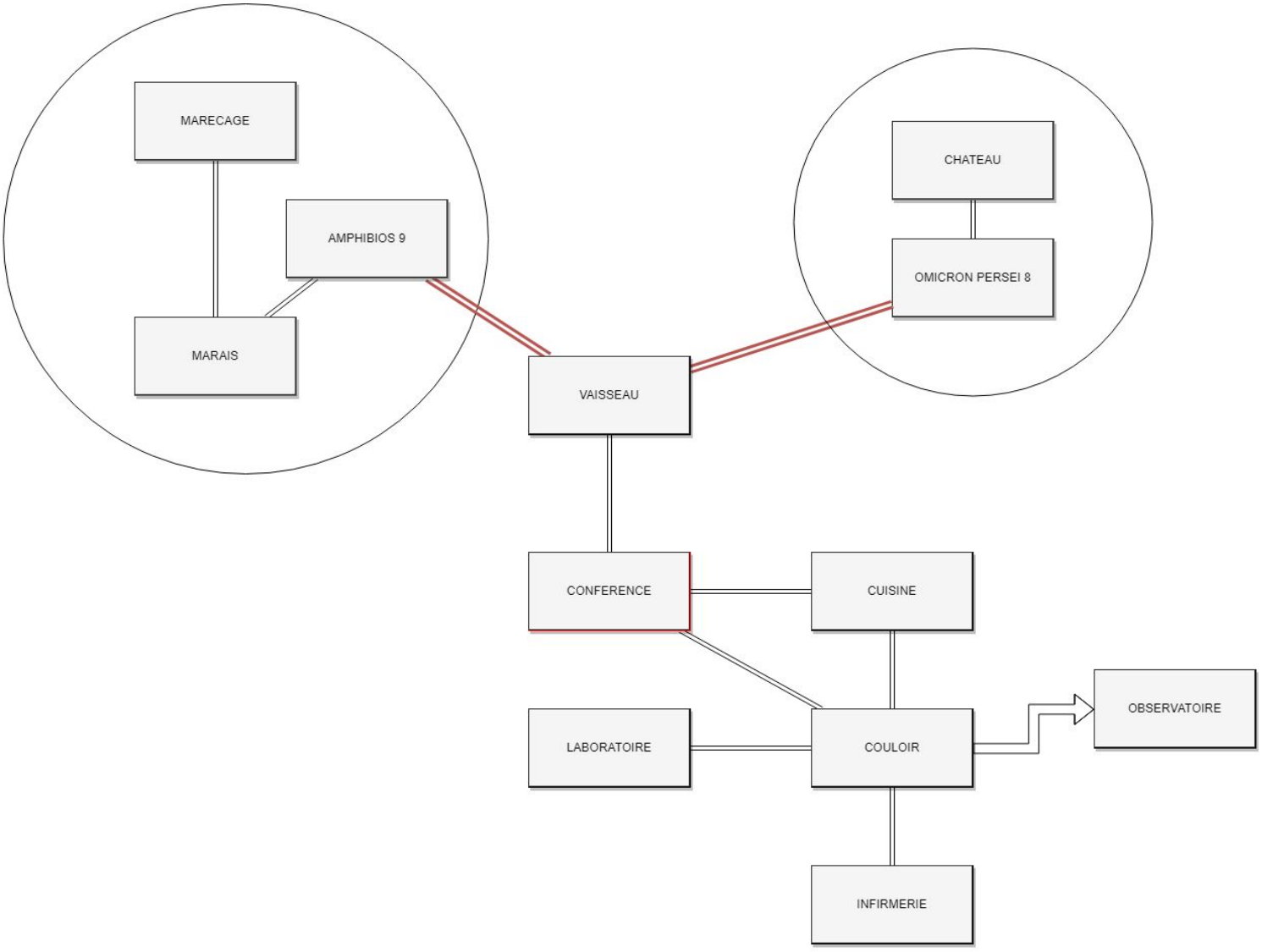
A la recherche du colis perdu

I.A) Auteur
Théo LEFÈVRE

I.B) Phase thème
Un jeune livreur doit retrouver le colis perdu et le remettre à son destinataire.

I.C) Résumé du scénario
Vous incarnez Fry, un jeune livreur, et vous devez partir à la recherche du colis perdus que dois acheminer l'équipe de Planet Express sur Omicron Persei 8.
Durant le jeu, vous serez amené à résoudre une enquête afin d'éviter la mort.

I.D) Plan



I.E) Scénario détaillé

L'action se déroule dans le futur. Planet Express, une société de livraison intergalactique a été chargée de livrer un colis pour le roi de la planète Omicron Persei 8.
Malheureusement, il s'avère que le colis a été égaré, et il ne serait diplomatiquement pas envisageable pour la Terre que ce colis ne soit pas livré.
Le jeu débute dans la salle de conférence de Planet Express, où le professeur annonce la mauvaise nouvelle. Fry (vous) et ses amis vont être chargés de retrouver ce colis et de le remettre sur Omicron Persei 8, évitant ainsi une nouvelle guerre entre cette planète et notre Terre.
Cependant, l'aventure ne sera pas de tout repos. Vous allez devoir chercher le colis dans les locaux de Planet Express, ainsi que sur deux planètes plus ou moins hostiles, où il aurait également pu être perdu.
Il s'avère qu'en cherchant le colis, vous en trouviez un deuxième. Comment savoir lequel est le bon ? Remettre le mauvais colis mettrait le roi fortement en colère, et votre vie serait finie.
Vous allez donc devoir mener une enquête en interrogeant les habitants à votre arrivée sur Omicron Persei 8. Les habitants vous donneront des indices, comme le contenue du paquet, etc, vous permettant de facilement faire le choix.
Attention à la planète Amphibios 9, elle est sauvage et un monstre s'y trouve. Vous aurez besoin de lunettes spécialement conçues par le professeur afin d'échapper à son emprise.

I.F) Détails des lieux, items, personnages.

<u>Le lieux :</u>

1) La Terre (locaux de Planet Express)
    - La salle de conférence : la salle principale, le professeur y annonce les modalités.
    - La cuisine : Fry peut y trouver de quoi manger, l'une des portes est piégée.
    - Le couloir : on y trouve bender qui peut nous être utile sur la route, de plus.
    - Le laboratoire du professeur : Fry peut y trouver les lunettes.
    - L'infirmerie : rien de spécial pour le moment. PNJ enquête ?
    - L'observatoire : Fry peut y voir les deux planètes. il y a un téléporteur.
    - Le vaisseau de Planet Express : permet de se téléporter sur les planètes.
    - La soute du vaisseau : elle contient un des colis.

2) Amphibios 9
    - Amphibios 9 (lieu d'atterrissage du vaisseau) : rien de spécial actuellement.
    - Un marais: rien de spécial, peut-être PNJ ?
    - Un marécage : il y a le deuxième colis, cependant Fry meurt s'il n'a pas les lunettes.

3) Omicron Persei 8
    - Omicron Persei 8 (lieu d'atterrissage du vaisseau) : PNJ pour enquête. Le chemin mène vers le château.
    - Le Château du roi : Remise des colis, fin du jeu.

Les personnages :

- Fry : le jeune livreur de Planet Express
- Bender : un robot capable de tordre n'importe quoi.
- Un monstre : il vous tue par hypnose.
- PNJ : un ou deux pour résoudre l'enquête.

Les items :

- Une part de Pizza (magic cookie): permet à Fry de ne pas mourir de faim.
- Des lunettes : elles sont fabriquées par le professeur, elle permettent de ne pas être hypnotisé par le monstre.
- Colis : deux, l'un est celui destiné au Roi.

I.G) Situations gagnantes et perdantes

Situation gagnante :

La partie est considérée gagnée lorsque le bon colis a été remis sur la bonne planète, et que Fry n'est pas mort.

Situation perdante :

La partie est considérée perdue si Fry meurt de faim ou par le monstre, ou si le mauvais colis est remis au Roi.

I.H) Énigmes, mini-jeux, combats

Enigme :

Dans le cas où vous auriez trouvé les deux colis, vous allez devoir enquêter afin de savoir lequel est le bon. Allez donc voir les PNJ sur Omicron Persei 8 pour cela !

I.I) Commentaires

Ce qui est fait :

- Les salles, les objets,
- La GUI et ses images / boutons,
- L'histoire.

Ce qui reste à faire :

- PNJ pour l'enquête.

II) Réponses aux exercices

7.5) printLocationInfo()

pas de réponse attendue, l'exercice porte sur la duplication de code dans une première partie, puis de la création de la procédure printLocationInfo() qui affiche la salle actuelle et ses sorties.

7.6) getExit()

pas de réponse attendue, on change le système des quatres directions pour une première approche d'un système évolutif qui permettra de définir les sorties en fonction d'une sortie passée en paramètre de la procédure getExit().

7.7) getExitString()

On définit une méthode qui va retourner la liste des sorties que l'on va placer dans la classe Room et que l'on appellera dans la procédure printLocationInfo() de la classe Game.
Il est logique d'écrire cette méthode dans la classe Room, car c'est dans la classe Room que va se trouver les sorties relatives à une salle. Il suffira donc à la procédure printLocationInfo() d'afficher ce que va lui retourner cette méthode sur un objet.

7.8) HashMap, setExit()

On définit la nouvelle façon de créer des Room. On utilise une nouvelle fonction qui dans l'exercice d'après utilisera une hashmap pour stocker les sorties. Lors de la définition des Room, au lieu de devoir définir quatre direction (null ou existante), il nous suffit d'appeler notre nouvelle fonction, à laquelle nous donneront le nom de la sortie, et la salle vers laquelle elle pointe.
La procédure inutile est donc à présent setExits car nous n'utilisons plus les quatres directions comme paramètres.
La notion de HashMap est maîtrisée, on associe des valeurs (direction) à des clés (salles).

7.9 & 7.10) keySet(), getExitString()

La méthode keySet() permet de voir les clés contenues dans la hashmap.
Nous allons ainsi utiliser cette méthode afin de retourner les clés associées à une salle, (donc ses sorties), puis les concaténer afin de les retourner sous forme d'une String.
Le for each peut se traduire par "pour toutes les occurrences", donc ici, pour toutes les clés associées à une sortie, concaténer leur valeur.
Ceci est réalisé dans la méthode getExitString().

7.11) getLongDescription()

Comme pour getExitString, on déplace ce qui concerne les Room dans la classe Room, et on affichera seulement ce que retourne cette méthode dans la classe Game.

7.14 & 7.15) look(), eat()

On a ajouté des commandes, on a d'abord déclaré qu'elles étaient valides dans la classe CommandWords, puis on a indiqué leur effet dans la classe Game.
Elles n'ont pour le moment pas d'effets particulier.

Question optionnelle : On procède de la même façon que pour la commande quit, la procédure associée à cette commande devra avoir un paramètre de type Command et un test de deuxième membre.

7.16 & 7.18) showAll(), showCommands(), getComandList()
On utilise ici une boucle for each afin de retourner toutes les commandes valides contenues dans le tableau de String de la classe CommandWords, puis on les concatènes dans la méthode showAll(). On utilisera la procédure showCommands() pour afficher la liste des commandes valides dans la classe Game.
On nommera showAll() en getComandList().

7.18.6) zuul-with-image
On modifie ici une grande partie de notre programme, on introduit un environnement graphique. L'interface graphique utilisateur est définie dans la nouvelle classe UserInterface. L'interface graphique peut se résumer en un Panel comprenant divers éléments comme l'image de la salle, la fenêtre de commande habituelle, des boutons, etc ..

7.18.8) Bouton
On créer un bouton de Type JButton, que l'on va ajouter à notre Panel graphique en lui précisant une région sur laquelle il sera fixé. On va ajouter un actionListener à ce nouveau bouton afin de pouvoir écouter et interpréter la commande attribuée au bouton.

7.19.2) Images
Pas de réponse attendue, on déplace simplement les images trouvées dans un Dossier Images, et on précise le chemin vers l'image dans la création des Room.

7.20) Item
On va créer une nouvelle classe Item afin de créer des attributs de types Item dans Room.

7.21) Item description
On redéfinit la fonction toString dans la classe Item afin qu'elle retourne contenant la description de l'item et son poids.

7.22) items
On va créer une collection d'Items (HashMap) dans la classe Room et une procédure permettant d'ajouter les Items à cette HashMap. On peut maintenant avoir une infinité d'Items.

7.22.2) Intégrer des objets
On utilise simplement notre procédure d'ajout d'Items dans la procédure createRoom() dans la classe GameEngine.

7.23 & 7.24 & 7.25) Back
On définit un nouvel attribut de type Room dans la classe GameEngine.
Cet attribut sera initialisé et redéfinit à chaque changement de salle, il prend la valeur de la salle dans laquelle nous nous trouvons précisément. On ajoute également la commande

Back et sa procédure associée, permettant de changer l'attribut CurrentRoom par la valeur de notre attribut Back. On ajoute également les tests pour les cas où l'on aurait pas changé de salle, on si un paramètre est ajouté à la fin de la commande back.
En changeant plusieurs fois de salle, on remarque que l'on ne peut revenir en arrière que d'une salle salle, ce qui est logique puisque l'attribue back ne peut enregistrer qu'une seule information, et pas le chemin parcouru.

7.26) Stack
On créer une nouvelle collection de type Stack (pile), qui va permettre d'enpiller les salles dans lesquelles nous sommes allés. On utilisera la procédure push() pour ajouter une salle à la pile, et pop() pour prendre la salle en haut de la pile, puis l'enlever de cette pile.
On testera si la pile est vide avec la procédure empty().

7.26.1) Javadoc
On génère les Javadoc utilisateur et programmeur à l'aide de deux commandes. Ces commandes vont nous retourner des Warnings ou Errors si la Javadoc a été mal écrite dans notre code, on corrigera ces erreurs pour avoir une Javadoc complète.

7.27) Tests
On peut vouloir tester les scénarios de notre jeu, sans avoir à taper nous même toutes les commandes pour y arriver !

7.28) Automatisation des tests
On va taper toutes les commandes nécessaires à exécuter dans un fichier texte, puis on va le lire à l'aide d'une procédure qui va scanner le fichier, puis exécuter ligne à ligne les commandes présentes dans ce fichier.
On utilise les exemples vus en TP afin de construire ce type de procédure.

7.28.1) Commande test
On réalise ce qui a été dit précédemment, un Scanner va ouvrir le fichier passé en paramètre s'il existe, exécuter les commandes, puis le fermer.

7.28.2) Fichiers de commandes
On créer des fichiers textes contenants des commandes du jeu afin de les tester.
On réalise un fichier qui réalisera le scénario du jeu.

7.29) Player
On créer une classe Player, qui sera chargée de contenir les fonctions propres au joueur, ceci afin d'alléger Game Engine. Cette classe implique que plusieurs joueurs peuvent être instanciés simultanément.
On définit un joueur par son nom, sa position dans le jeu, et la charge maximum d'objets qu'il pourra porter. On va également gérer les Items dans cette classe, car en effet les objets seront propres au joueur. De même, la position du joueur sera enregistrée dans cette classe.
La procédure back sera donc en partie gérée par cette classe (la pile de positions).

Player étant le nouveau fonctionnement du jeu, un ajoute un attribut player dans GameEngine, et on créer un nouvel utilisateur dans le construction de GameEngine.

7.30) take(), drop()

On va définir les fonctions take et drop telles qu'elles permettront de mémoriser un unique attribut, désignant l'Item que l'on désire prendre ou jeter.

7.31) Porter plusieurs Items

On va modifier les procédures take et drop de telles sortes à ce qu'elles manipulent des HashMap d'Item.

On place également un Item dans la pièce principale.

7.31.1) ItemList

On créer une nouvelle classe ItemList, qui va se charger de gérer l'inventaire du joueur et des objets des Room.

Cette classe manipule essentiellement une HashMap, elle comprendra donc fonction permettant de manipuler cette HashMap (ajouter ou supprimer des éléments).

7.32) Poids max

On ajoute un attribut poids max dans la classe Player, ainsi qu'une méthode permettant de tester si un joueur peut encore porter des items ou non en faisant la somme du poids du joueur et de l'objet qu'il transporte.

7.33) Inventaire

On ajoute la commande inventaire qui appelle la méthode inventaire. Elle va deja tester si l'inventaire est vide, et ensuite elle appellera une méthode déjà définie qui retourne la liste des objets de l'inventaire sous forme de String.

7.34) Magic-Cookie

On modifie la procédure eat pour qu'elle puisse reconnaître si le paramètre correspond au Magic-Cookie, ensuite, on double le poids maximal que peut porter l'utilisateur.

On fait ensuite disparaître le Cookie du jeu.

7.42) Time Limit

On créer un attribut qui va compter le nombre de commandes de déplacement dans la classe GameEngine, à chaque appel de la commande de déplacement, un compteur incrémente. Lorsque l'on effectue 10 déplacements d'affilés sans manger, le jeu s'arrête.

7.42.2) IHM

Exercice non traité.

7.43) Trap Door

On modifie la structure du constructeur de la classe Room pour qu'il prenne un paramètre boolean supplémentaire. Lors de la création d'une Room, on choisira false ou true si les portes associées à cette salle sont piégée ou non.

Si une porte est piégée, l'utilisation de la commande back ne sera pas possible.

7.44) Beamer
On créer une nouvelle classe Beamer qui hérite de Item. Elle contiendra les méthodes de chargement du beamer, ainsi que des accesseurs aux attributs boolean et de type Room de la classe.
Dans GameEngine, on créer deux méthodes charge() et use() prenant en paramètre le nom de l'item associé au beamer. Le chargement permettra de prendre en mémoire la salle que l'on souhaite sauvegarder pour une éventuelle future téléportation. L'utilisation aura pour but de nous déplacer vers cette salle sauvegardée depuis une autre salle du jeu, de décharger le beamer et de vider la pile de back.

7.46) Transporter Room
Exercice compris mais non réalisé.
Il faudrait créer deux nouvelles classes :
Une classe TransporterRoom qui hérite de Room, qui permettrait de créer les Room spéciales
Une classe Randomizer, qui contiendrait un attribut allRooms de type HashMap, contenant la liste complète des Room du jeu, et qui par une fonction appelant la classe Random de Java, sélectionnerai une salle au hasard parmi cette liste.
On associe une pièce à un numéro à l'aide de la HashMap, et grâce à une fonction Random, on va tirer un nombre aléatoire.

7.46.1) Commande aléa
Exercice compris mais non réalisé.
Avec la commande alea String, on va choisir la prochaine salle vers laquelle nous redirigera la transporter room, et la commande alea sera chargée de vider la salle passée en mémoire par la commande précédente.

7.46.2) Héritage
Exercice non traité, mais principe compris

III) Déclaration anti-plagiat :
La plus grande majorité de mon code m'appartient, cependant, je reconnais avoir demandé des conseils aux enseignants qui m'ont guidé pour réaliser les commandes take et drop notamment.