Searching for the slime breeder

lien page web
(IPO 2020/2021 G12)

I.A) Auteur(s)

Brendan VICTOIRE groupe 12

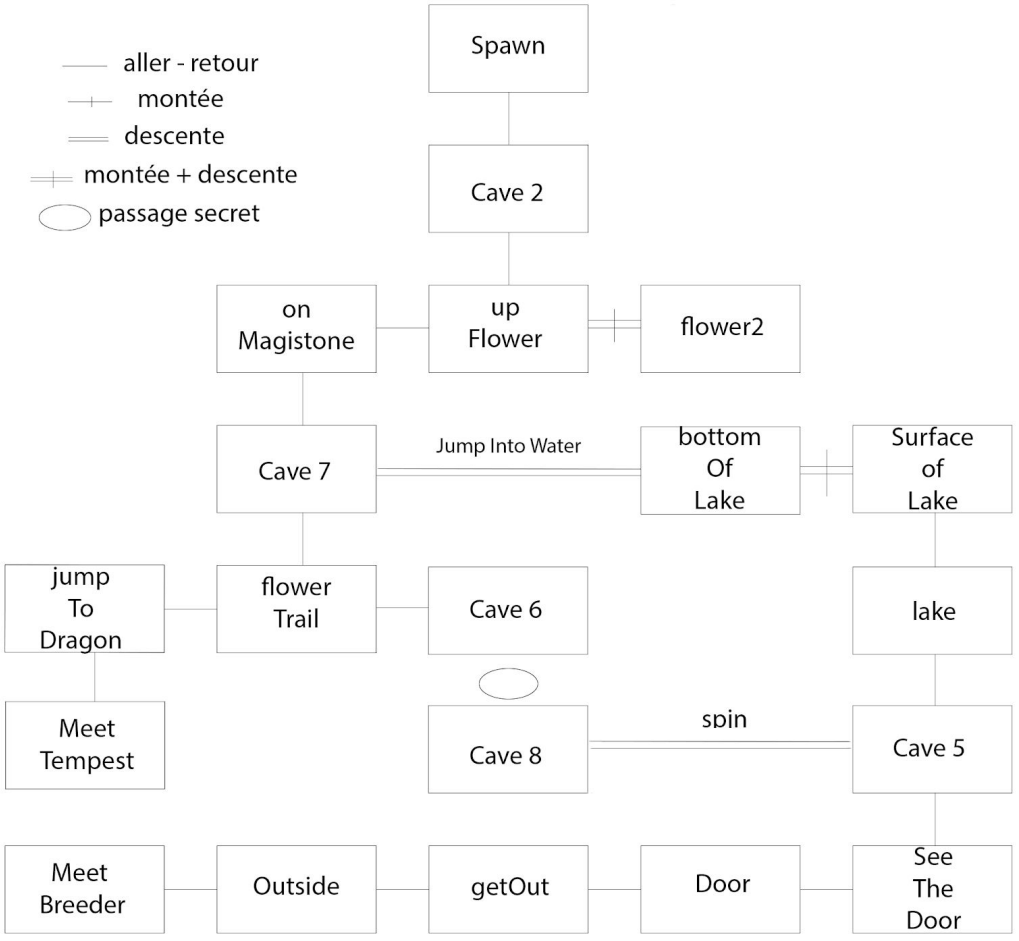
I.B) Thème

Un slime à la recherche de son éleveuse

I.C) Résumé du scénario

Démarrage dans une grotte et essayé de trouver la sortie pour retrouver l'éleveuse

I.D) Plan



I.E) Scénario détaillé

Vous êtes un slime perdu dans une grotte à la recherche de son éleveuse. Vous devez rencontrer Veldora pour accéder à la porte de sortie

I.F) Détail des lieux, items, personnages

Lieux : - la grotte et ses différentes salles, l'extérieur de la grotte

Items : Magistone, une fleur, l'âme du dragon, une pierre

Personnages : Rimuru (le slime), Veldora (le dragon), Beatrix LeBeau (l'éleveuse)

I.G) Situations gagnantes et perdantes

Situation gagnante : retrouvé Beatrix;

Situation perdante : - ne pas prendre l'âme de Veldora ou la manger;

- ne pas rencontre Veldora;
- prendre trop de temps;
- sauter dans le lac sans voir absorbé l'âme de Veldora

I.H) Éventuellement énigmes, mini-jeux, combats, etc.

Lors de la rencontre avec Veldora essayé de prendre 2 fois son âme pour y parvenir

Dans la cave6 avoir l'âme de Veldora pour découvrir une porte cachée

I.I) Commentaires (ce qui manque, reste à faire, ...)

Accès au raccourci

II. Réponses aux exercices **(à partir de l'exercice 7.5 inclus)**

7.5 Remplacement des parties de code qui étaient identique dans `goRoom()` et `printWelcome()` par l'appelle d'une procédure `printLocationInfo()` créer pendant l'exercice.

7.6 Création d'un accesseur pour les sorties qui est ensuite appelé dans `goRoom()` et `printLocationInfo()`.

7.7 Création d'un accesseur pour récupérer les sorties de la salle actuelle sous forme de string ce qui amène une modification de `printLocationInfo()`.

7.8 Utilisation de HashMap pour créer une fonction pour modifier les sorties des salles + création des sorties des salles.

7.8.1 Rajout des directions "`up`" et "`down`".

7.9 Création d'une variable mémoire des directions dans `getExitString()` pour pouvoir afficher les sorties disponibles.

7.10 On a créé une variable de type Set qui contient l'ensemble des directions puis grâce à une boucle for Each on compose le String avec les sorties possible.

7.10.1 Ajustement des commentaires.

7.10.2 Génération du Javadoc.

7.11 Ajout d'une méthode `getLongDescription()` pour avoir une description détaillée et modification de `printLocationInfo()` grâce à cette méthode.

7.14 Ajout d'une méthode `look()` qui affiche la description longue de la pièce actuelle si la commande n'a pas d'instruction.

7.15 Ajout d'une méthode `eat()` qui affiche le fait d'avoir mangé si la commande a une instruction.

7.16 Ajout de méthodes dans les classes `CommandWords` et `Parser` pour pouvoir afficher toutes les commandes dans le message d'aide la classe `Game`.

7.17 La classe `Game` n'a plus besoin d'être modifié vu que l'on a créée des méthodes dans l'exercice d'avant qui affiche les commande de l'attribut de la classe `CommandWords`.

7.18 modification des procédures `showCommands()` et `showCommandList()` par des méthodes renvoyant des String.

7.18.1 Rien à modifier.

7.18.2 Remplacement des String par des `StringBuilder` dans `Room.getExitString()`

7.18.3 Photo déjà cherché et trouvé dès le début du projet.

7.18.4 Titre déjà choisi dès le début du projet.

7.18.5 Création d'une `Hashmap` pour contenir l'ensemble des pièces.

7.18.6 Modification de classes pour pouvoir rajouter des images pour chaque salle et changer l'interface graphique du jeu.

7.18.7 ¤ à faire ? ¤

7.18.8 Ajout de 13 boutons dans la classe `UserInterface` avec modification des procédures `actionPermformed()` - pour que les boutons exécutent la commande qui leur est associée - et de `createGUI()` - pour afficher le bouton et le rendre utilisable.

7.19 ¤ à faire ? ¤

7.19.1 ¤ à faire ? ¤

7.19.2 Ajout d'image pour le jeu et affichage de celle-ci.

7.20 Ajout d'une classe `item` pour créer des items avec leur nom, description et poids, ainsi que des accesseurs pour chacun des attributs et une fonction `toString()` pour potentiellement afficher l'item

7.21 Ajout d'une méthode `getItemString()` dans `Room` pour pouvoir afficher le nom de tous les potentiels items de la room actuelle avec la méthode
Ajout d'une méthode `setItem()` pour pouvoir ajouter une item a l'ensemble des items de la room

`getLongDescription()` et ajout d'une méthode `hasItem()` dans `Room` pour pouvoir ne rien afficher avec `getItemString()` si la room actuelle n'a pas d'item.

7.21.1 Modification de `look()` pour que si le instruction de la commande correspond à un item, l'item `toString()` soit affiché.

7.22 Déjà fait durant l'exercice 7.21 (juste renommage de `setItem()` en `addItem()`).

7.22.1 - `LinkedIn` n'est pas pratique, car les objets sont ranger et donc si un objet est supprimé de la liste alors tous ceux qui suivent verront leur index changer ce qui n'est pas très pratique. De plus s'il y a beaucoup d'objets le temps de réponse augmente proportionnellement à l'index de l'objet demandé. Et enfin on ne peut vérifier que la référence de l'objet et non l'existence exacte de celui-ci

- `ArrayList` se redimensionne à chaque fois que l'on ajoute un objet si la taille prévue de base n'est pas suffisante ce qui entraîner des ralentissement.
- `HashTable` renvoie un entier, ce n'est pas ce que nous voulons
- `HashSet` obligation de créer un `Iterator` pour parcourir le `Set`

Ce qui explique le choix de `HashMap` qui comble les problèmes listé au-dessus.

7.22.2 Ajout d'item à des room.

7.23 Ajout d'une procédure `goBack()` pour retourner à la salle précédente.

7.24 Fait dans l'exercice 7.26 grâce à `stack`.

7.25 Fait dans l'exercice 7.26 grâce à `stack`.

7.26 Modification de l'attribut correspondant à la salle précédente pour qu'il corresponde à l'ensemble des salles précédentes et par conséquent de `goBack()` pour prendre en conte le `stack`.

7.26 Génération du Javadoc.

7.28.1 Création d'une procédure `test()` qui entraine la modification de `interpretCommand()` et de l'attribut de `CommandWords` et qui permettra de vérifier

qu'après les modifications du code dans les futurs exercices les commandes fonctionnent toujours correctement. Ces tests se font grâce à l'exécution d'un d'un fichier `.txt` dans lequel nous avons écrit au préalable les commandes à tester.

7.28.2 Création de fichier `.txt` pour tester les commandes et les salles.

7.29 Création d'une classe `Player` dans laquelle nous avons déplacé. `aCreateRoom` et `aPreviousRooms` et ajouté des accesseur et modificateur pour ceux-ci. Mais aussi dans `Player` nous avons déplacé toutes les procédures qui permette d'exécuter une commande (sauf pour `test()`).

7.30 Création de procédure `take()`, `drop()` et de procédure permettant le fonctionnement de celle-ci et d'éviter le doublon de code dans `Player`.

7.31 Déjà fait précédemment grâce à `hashMap`.

7.31.1 Création d'une classe `ItemList` pour gérer les items de `room` et `player`. Ceci implique la suppression des méthodes dans `room` et `player` qui s'appelle sur un item pour les réécrire dans `ItemList`.

7.32 Création d'un attribut `MAX_WEIGHT` pour limiter la charge que peut transporter le personnage et création par conséquent d'une méthode pour savoir connaitre le poids de la charge transporter par le personnage.

7.33 Création d'une commande `items` qui affiche la liste des `items` transporté par le joueur.

7.34 Modification de la méthode `eat()` pour pouvoir augmenter le poids de la charge que peut transporter le personnage après avoir mangé `"dragonSoul"`.

7.34.1 Modification des fichier de test pour prendre en compte les nouvelle commandes et possibilité.

7.34.2 Génération du Javadoc

.

7.35 Création d'une enum `CommandWord` pour implémenter la possibilité de jouer dans différente langues. Et modification par conséquent de `InterpretCommand()`, du `Parser`, de `CommandWords` pour inclure les `Command` sous forme d'enum.

7.35.1 Modification de la suite de `if else{}` dans `interpretCommand()` par un `switch{}`.

7.37 Le jeu reste en anglais donc rien à modifier.

7.38 Le jeu reste en anglais donc rien à modifier.

7.40 ¤ comment rédiger ça ¤

7.41 ¤ comment rédiger ça ¤

7.41.1 Incorporation de *zuul-with-enums-v2*.

7.41.2 Génération du Javadoc.

7.42 Ajout dans `Player` d'un attribut `aCommandEntered` pour conter le nombre de commandes effectuer par le joueur et le faire quitter le jeu de force si la limite a été dépassé.

7.42.1 ¤ à faire ¤

7.42.2 Pour le moment, je me contente de l'interface graphique actuelle (en ayant rajouté quelque bouton).

7.43 Déjà pris en compte depuis le début du jeu donc pas de modification à faire.

7.44 Ajout d'une classe `Beamer` qui est une extension de Item et qui permet de mémoriser (`memorize()`) une room pour s'y téléporter (`teleport()`) par la suite mais a usage unique. Implémentation de boutons correspondant a ces deux commande. Et enfin mis à jour des fichier test pour tester le bon fonctionnement de `Beamer`.

7.45 Création d'une classe `Door` pour permettre de bloquer des passages et implémentation de `createRooms()` et de `goRoom()` grâce cette classe. Et mis a jour des fichier test pour le bon fonctionnement des tests.

7.45.1 Déjà fait dans les exercices précédents.

7.45.2 Génération du Javadoc.

III. Mode d'emploi (si nécessaire, instructions d'installation ou pour démarrer le jeu)

IV. Déclaration obligatoire anti-plagiat (préciser toutes les parties de code que vous n'avez pas écrites vous-même et citez la source, sauf les fichiers zuul-*.jar qui sont fournis évidemment)

Toutes les images sont tirées des deux premiers épisodes de *Tensei shitara Slime Datta Ken* sauf la rencontre avec l'éleveuse tirée du jeu Slime rancher + le slime de *Tensei shitara Slime Datta Ken.*