Itération 2 du Jeu d'aventure

- *Lire le chapitre 7 jusqu'à l'exercice 7.18 non inclus.*
- Faire l'exercice **7.18** (getCommandList).
- Exercice **7.18.1** : Comparer son projet à celui-ci (gestion des sorties et des descriptions du projet *zuul-better*; corriger si besoin est, **sans régression** bien sûr ! (notamment, pas d'itérateur quand on peut utiliser une boucle *for each*) ; tout recompiler : il ne doit y avoir aucun warning.
- Exercice **7.18.2** : Étudier la documentation de StringBuilder pour comprendre son utilité et l'intégrer à son jeu, **notamment** dans `Room.getExitString()`.
- Apprentissage: `StringBuilder`
- Exercice **7.18.3** : Commencer à chercher une image différente pour chaque `Room`. Ce peut être une photo, une image de synthèse, un dessin scanné, ...
- Exercice **7.18.4** : Décider du titre du jeu, et l'incorporer dans le message de bienvenue du jeu.
- Exercice **7.18.5** : Pour avoir accès aux objets `Room` depuis n'importe quelle classe, créer une `HashMap` contenant toutes les Room (associées à leur nom). Il suffira alors de la passer en paramètre, en cas de besoin.
- Exercice **7.18.6** : Étudier le projet suivant (Game, GameEngine, UserInterface) : du projet *zuul-with-images* pour comprendre son fonctionnement global et intégrer dans son jeu (**sans régression** bien sûr ! notamment en ce qui concerne la généricité des collections : il ne doit subsister aucun warning/avertissement !) cette nouvelle conception qui permettra d'opter éventuellement par la suite pour une interface graphique plus élaborée.
- Exercice **7.18.7** : Décrire le fonctionnement de `addActionListener()` et `actionPerformed()` dans `UserInterface`.
- Exercice **7.18.8** : Ajouter au moins un bouton qui déclenche une des commandes du jeu.
- Apprentissage : `ActionListener`, `addActionListener()` et `actionPerformed()`
- Faire l'exercice **7.19** (MVC).
- Exercice **7.19.1 optionnel** : Après avoir fait l'exercice 7.19, étudier le projet suivant (Game, GameModel, TextView) : projet *zuul-mvc* pour comprendre son fonctionnement et intégrer dans la version "with-images" que l'on vient de créer cette nouvelle conception plus proche de l'architecture MVC qui permettra par exemple de faire cohabiter 2 observateurs (texte + graphique).
- Exercice **7.19.2** : Incorporer dans le jeu une image différente pour chaque `Room`. S'il en manque, fabriquer une "image de mot" pour afficher simplement le nom du lieu.
- *Lire le chapitre 7 jusqu'à l'exercice 7.20 non inclus.*
- Faire l'exercice **7.20** dans son jeu (Item); il est possible de remplacer le poids de chaque item par un prix, ou même de prévoir les deux !
- Faire l'exercice **7.21** (item description **sauf** Explain in writing); "information about" est à prendre au sens de "toutes les informations sur".

Il est également possible de prévoir une `longDescription` pour chaque `Item` ; voir quand il serait intéressant de l'utiliser.

- *Lire le chapitre 7 jusqu'à l'exercice 7.22 non inclus.*
- Faire l'exercice **7.22** (items).
- Exercice **7.22.1 optionnel** : Justifier par écrit le choix du type de collection utilisé à l'exercice 7.22.
- Exercice **7.22.2** : Intégrer les objets (items) de son jeu (au moins ceux du sous-scénario).
- *Lire le chapitre 7 jusqu'à l'exercice 7.23 non inclus.*

- Faire l'exercice **7.23** (back).
- Faire les exercices **7.24** (back test) et **7.25** (back back). Modifier son jeu s'il n'a pas le comportement désiré.
- Faire l'exercice **7.26** (Stack). La commande `back` fonctionne-t-elle si l'on est revenu au point de départ du jeu ? Modifier son jeu s'il n'a pas le comportement désiré.
- Apprentissage: Stack, push(), pop(), empty(), peek()
- Exercice **7.26.1** : Générer les 2 javadoc du projet en utilisant :

```
javadoc -d docprog -author -version -private -linksource *.java
javadoc -d docuser -author -version *.java
```

à partir de la ligne de commande.