L'implantation complète de tous les changements présentés jusqu'ici est disponible par l'intermédiaire du projet nommé *zuul-better*. Si vous avez effectué les exercices jusqu'à présent, vous pouvez ignorer ce projet et continuer avec le vôtre. Si vous n'avez pas traité ces exercices, mais souhaitez effectuer les prochains exercices du chapitre à titre de projet de programmation, vous pouvez utiliser le projet *zuul-better* comme point de départ.

Prévoir les évolutions futures

L'architecture des classes que nous avons implantée est une amélioration importante de la version originale. Cependant, il est encore possible de l'améliorer.

Une caractéristique d'un bon concepteur logiciel est sa capacité à prévoir les évolutions futures. Qu'est-ce qui pourrait changer ? Quelles sont les parties dont nous sommes sûrs qu'elles resteront inchangées durant toute la vie du programme ?

Une hypothèse que nous avons codée en dur dans la plupart de nos classes est que le jeu sera exécuté à l'aide d'une interface textuelle en entrée et en sortie. Mais en sera-t-il toujours ainsi ?

Il peut sembler intéressant de proposer plus tard une interface graphique qui contiendra des menus, des boutons et des images. Dans ce cas, nous ne voudrons plus afficher les informations sous forme textuelle à l'aide d'un terminal. Nous pourrions toujours avoir des commandes données sous forme de mots, et nous pourrions toujours vouloir les afficher au moment où l'utilisateur saisit une commande d'aide. Cependant, nous pourrions les afficher alors dans le champ texte d'une fenêtre graphique plutôt que d'utiliser `System.out.println`.

Pour obtenir une application bien conçue, il faut essayer d'encapsuler toutes les informations de l'interface utilisateur au sein d'une classe (ou bien un ensemble bien défini de classes). Par exemple, la méthode `showAll` de la classe `CommandWords` de la solution proposée précédemment dans la section « Couplage implicite » ne respecte pas cette règle de conception. Il serait plus intéressant de définir la classe `CommandWords` comme chargée de la *production* (et non de l'*affichage*) de la liste des commandes, la classe `Game` devrait alors décider de la manière de la présenter à l'utilisateur.

Nous pouvons facilement réaliser ce changement en modifiant la méthode `showAll` de telle façon qu'elle renvoie une chaîne de caractères contenant toutes les commandes plutôt que de les afficher directement (nous devrons probablement renommer cette méthode `getCommandList` quand nous ferons ce changement). La méthode `printHelp` de la classe `Game` pourra alors afficher cette chaîne.

Cette modification n'apporte aucun avantage jusqu'à maintenant, mais cette amélioration pourrait néanmoins se révéler profitable à l'avenir.