Projet de POCA — Nethack

Version 2.0

Université Paris Diderot – Master 2 SRI/LC/LP

12 décembre 2012

1 Principe du projet — rappel

Le projet de POCA vise à vous faire progresser dans votre capacité *à concevoir un système objet extensible*.

Il se déroule en *deux étapes*. Dans la première étape, un sujet vous est fourni. Ce sujet est *volontairement* décrit de façon informelle, sans vous donner d'indication pour le réaliser. Vous devez donc définir vous-même une *architecture* pour votre projet et vous la décrirez dans le **délivrable 1**.

Une *implémentation* écrite en Scala de cette première version constitue le **délivrable 2**.

Vous recevrez ensuite un *autre sujet* (celui-ci) qui est une extension du premier. Cette extension sera l'objet du **délivrable 3** et mettra à l'épreuve votre architecture initiale : un projet bien pensé *n'aura pas besoin de modifier* le code de la première version du projet pour traiter cette extension mais seulement de *rajouter de nouveaux composants*.

2 Sujet

- une première généralisation consiste à **ajouter les personnages, types de monstres, et objets** de NetHack qui vous n'aviez pas inclus dans la première version. Le but ici est de convaincre (vos enseignants et vous-mêmes) de la possibilité de les ajouter sans changer la structures de classes, mais simplement en ajoutant des nouvelles classes.
  Quels sont les personnages, monstres, et objets les plus difficiles à ajouter ?
- une deuxième généralisation est plus technique. Vous avez dévelopé votre jeu avec un des *toolkits* graphiques au choix entre Swing, Java Curses, Java GNOME, Qt Jambi. Maintenant vous devez **supporter un autre toolkit graphique** et permettre, au moment de l'exécution, de choisir entre les deux.
  Combien des classes avez-vous dû toucher pour réaliser cette modification ?
- une troisième généralisation consiste à autoriser la **création de dongeons infinis**, qui se construisent au fur et à mesure qu'on les visite.

- une quatrième généralisation est le **voyage dans le temps**. Certains sorts renvoient le joueur *à un instant choisi aléatoirement* dans le passé.
- changement de casquette : **jouer comme un monstre**. Avec une commande spécifique pendant le jeu, le joueur peut "tricher" (ou s'amuser) en décidant de prendre le rôle d'un monstre et faire agir son ancien joueur par une intelligence artificielle.
  Est-il simple d'adapter l'intelligence artificielle préexistant (p.ex. pour les monstres) au rôle du joueur ?
- plus difficile, vous devez changer le mécanisme d'interaction entre jeu et joueur pour passer du style tour-par-tour à un style **temps réel**, ou les joueurs et le monstre sont actifs à chaque instant (donc si le joueur ne fait rien, il sera probablement tué par le premier monstre qui passe...).
- ***hot seat*** : vous voulez permettre à plusieurs joueurs humains de jouer ensemble, tour-par-tour, chacun avec son propre personnage.
- **déploiement *client-server*** : une fois réalisé la possibilité de jouer à plusieurs, il est naturel de distribuer le jeu en réseau, avec un serveur de jeu qui garde l'état et plusieurs clients qui offre l'interface utilisateur et parlent avec le serveur.
  Est-il possible et simple d'utiliser vos classes pour compiler plusieurs exécutables, un pour le serveur, et un pour le client ?

On vous propose donc plusieurs extensions, vous devrez réfléchir à la réalisation de toutes ces extensions, et en réaliser le plus possible sous forme de code Scala.

3 Travail demandé

L'utilisation du SVN est obligatoire. L'historique de ce dernier permettra de déterminer la contribution des membres de l'équipe et la gestion du temps dont vous avez fait preuve.

Avant le 12 novembre 2013

Dans un répertoire `delivrables/architecture/` de votre SVN, vous devez nous soumettre une architecture sous la forme d'un ou plusieurs diagrammes de classes UML et d'autres diagrammes de votre choix accompagnés d'explications justifiant vos choix.

Ce document sera fourni au format PDF et pourra suivre le plan suivant :

interprétation du sujet Vous expliquerez de façon informelle ce que vous avez compris du sujet et de ces enjeux. Quels sont les problèmes techniques et conceptuels que vous avez exhibés ?

concepts Dans cette section, vous définirez les concepts utilisés pour modéliser le problème ainsi que les invariants essentiels du système.

description de l'architecture Vous donnerez ici les diagrammes UML décrivant votre architecture et surtout sa justification.

extensions envisagées Vous énumérez ici les généralisations et les extensions que vous avez imaginées et vous expliquerez pourquoi votre architecture permet de les traiter facilement.

Avant le 10 décembre 2012

Dans un répertoire `delivrables/version1/` de votre SVN, vous devez soumettre une première version de votre système.

Celui-ci doit évidemment compilé à l'aide d'une commande `make` effectuée à la racine de ce répertoire. Un fichier README doit être fourni aussi et doit expliquer comment exécuter votre système.

Votre projet doit aussi être testé. La qualité du code—c'est-à-dire sa correction, sa robustesse et son élégance—sera prise en compte dans la notation.

Avant le 14 janvier 2012 Dans le répertoire `delivrables/version2/` du SVN, vous déposerez votre implémentation de la version 2 du projet. Celui-ci doit évidemment compiler à l'aide d'une commande `make` effectuée à la racine de ce répertoire. Votre projet doit aussi être testé. La qualité du code—c'est-à-dire sa correction, sa robustesse et son élégance—sera pris en compte dans la notation. Enfin, l'ensemble des différences entre la version 1 et la version 2 devront être résumées dans un fichier nommé `CHANGES` que l'on trouvera à la racine de la version 2 du projet.