Départ. Informatique
LP DEVOPS - Module Conception / Méthodes Agiles

V. Deslandres

Fiche d'exercices – Rappels UML –

Exercice 1 – Croisement de routes (DSQ, DCL)

On suppose développer un simulateur du trafic routier d'une ville, qui fonctionne avec un pas de temps donné. On gère des carrefours de taille différente : ils ont des routes et des voies en nombre variable. Ces carrefours sont munis de feux de circulation. Un feu est identifié par ses coordonnées GPS, et est temporisé par une horloge.

2. 1- Décrire par un **diagramme de séquences** du mécanisme de lancement d'une simulation : au démarrage de la simulation par l'utilisateur, le système crée par ex. 2 feux, chacun avec leur horloge. L'utilisateur définit ensuite son pas de temps, puis lance la simulation : le simulateur se charge alors de mettre à jour l'affichage des horloges à chaque avancée du temps simulé.

2. 2- Donner le **diagramme de classes** correspondant.

Exercice 2 – Jeu AkinatorAnimal

Le but du jeu est de réaliser un jeu *AkinatorAnimal* permettant de faire deviner au logiciel, un animal pensé par l'utilisateur.

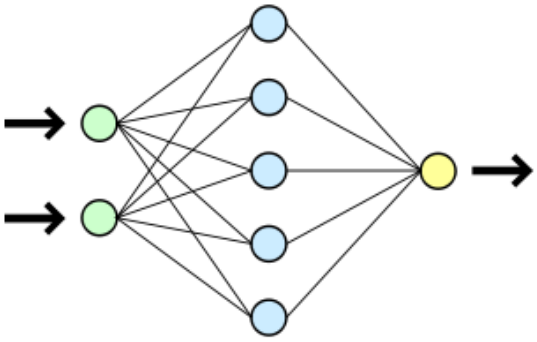
L'utilisateur choisit un animal, et l'application pose des questions auxquelles l'utilisateur peut répondre par une des 5 réponses suivantes : oui, probablement oui, ne sais pas, probablement non, et non. En répondant aux questions, vous permettez au logiciel d'éliminer des ensembles de réponses possibles, mais vous contribuez également à définir votre animal pour les parties suivantes.

Au bout d'une série de 10 questions, l'ordinateur propose sa réponse. L'utilisateur confirme ou infirme la réponse de l'ordinateur. Si la réponse est fausse, l'ordinateur demande à l'utilisateur s'il souhaite repartir pour une autre série de questions. Au bout de 3 séries, si le jeu n'a pas deviné votre animal, il admet avoir perdu et demande à l'utilisateur la bonne réponse parmi une liste de réponses susceptibles de convenir. Il l'entre alors dans sa base de données, se nourrissant ainsi des réponses que vous avez données.

A la fin d'une partie, qu'elle soit gagnante ou perdante pour l'ordinateur, celui-ci demande à l'utilisateur s'il souhaite ajouter une question pertinente à propos de son animal, et propose ensuite à l'utilisateur d'y répondre pour 10 autres animaux proches, se trouvant déjà actuellement dans sa base de données. Les propositions des utilisateurs sont évaluées par un modérateur avant d'être ajoutées à la BD. C'est ainsi que les joueurs enrichissent eux-mêmes la base de données. Le modérateur balaie régulièrement le contenu des BD et les nettoie. Ainsi les personnages qui ne sont pas joués suffisamment finissent par être éliminés automatiquement.

En fin de partie gagnée par l'application, le système affiche le nb de fois où l'animal a déjà été joué, et fournit la date et heure de la dernière partie concernée.

Le joueur a aussi la possibilité, une fois la partie achevée, de revoir toutes les questions posées, les réponses qui ont été données par le joueur et celles normalement attendues pour l'animal (rapport de partie). Il peut aussi, à chaque question, corriger la réponse de la question précédente.



Pour info : Akinator fonctionne sur un moteur créé sur mesure, nommé Limule, qui est écrit en C++. Sa base de données MySQL contient environ 100 000 personnages. Le fonctionnement de Limule s'appuie sur un système de réseau de neurones. Le serveur du jeu repose sur 15 serveurs pour permettre aux 250000 joueurs quotidiens d'accéder rapidement à la BD sans problème.

L'utilisateur n'a pas besoin de créer de compte pour jouer. Dans 85% des cas, le jeu devine en 2 séries (20 questions).

Travail à faire :

1- Élaborer le **Diagramme des Cas d'Utilisation** de ce jeu

2- Donner le SCN Nominal du **Cas « Lancer une partie »** et imaginer les exceptions

3- Construire le **Diagramme d'Activités du Cas « Lancer une partie »**

4- Construire le **Diagramme de Classes**

5- Élaborer le **Diagramme de Séquences** du scenario où la partie est gagnée par *AkinatorAnimal*.