Explications et justifications des choix de conceptions

Audibert Julien, El Hajami Mehdi, Finkelstein Arthur, Limam Mohamed

Table des matières

Informations générales :

Nous avons utilisé comme base du simulateur le rendu du TP long précèdent avec des ajouts mineurs de fonctionnalité.

Utilisation du maven :

Le projet a de base les plugins dans le fichier src, afin de facilité son utilisation dans Eclipse, mais il suffit d'utiliser le script deploy.sh pour mettre tous les fichiers des plugins dans le dossier myplugins/repository. Un script faisant l'inverse existe et s'appelle undeploy.sh.

Pour lancer le projet, il suffit de faire un « mvn clean package », suivi d'un « mvn exec:exec ».

La commande d'exécution peut prendre un argument pour lancer un scénario d'utilisation, en utilisant l'argument « –DdemoTest= » suivi du nom d'un fichier de config sans l'extension.

Le panneau de configuration est surchargé, les infos ne sont donc pas à jour lors d'utilisation d'un scénario.

Exemple : *mvn clean package ; mvn exec:exec -DdemoTest=Immortels (les autres noms des config se trouve ci-dessous).*

Utilisation des fichiers de config :

Les différents fichiers de config permettent de sélectionner différents paramètres présents dans le domaine et lancer la simulation choisie paramétrable ou non. Pour cela les fichiers de config contiennent des selects permettant la sélection de différents paramètres dans le fichier liant le domaine et l'implémentation. Il y a 7 fichiers différents de configuration :

1) Immortels *-> Démo non paramétrable : Avec Peu de créatures, comportement troupeau, snapshot long avec 10 points d'énergies de taille 50.*

2) MortIneluctable *-> Démo non paramétrable : 1 seule créature avec 1 point d'énergie de taille 2000, déplacement rebond, snapshot long.*

3) ModeleRecherche *-> Démo semi-paramétrable : Comportement recherche 5 points d'énergies de taille 50 avec peu de créatures, snapshot moyen.*

4) ModelePredateur *-> Démo semi-paramétrable : Comportement prédateur avec beaucoup de créatures en temps réel lent.*

5) BcpEngSrc *-> Démo entièrement paramétrable avec Beaucoup de points d'énergie : Peu de créatures déplacement rebond en mode debug.*

6) PeuEngSrc *-> Démo entièrement paramétrable avec Peu de points d'énergie : Beaucoup de créatures comportement prédateur snapshot court.*

7) PasEngSrc *-> Démo entièrement paramétrable avec aucun point d'énergie : Beaucoup de créatures comportement troupeau en mode temps réel très rapide.*

Features models :

Les features models sont dans le dossier src/commons et le fichier se nomme simu.fml

Implémentation :

Le feature model d'implémentation va essayer de coller le plus près au code possible. Ainsi nous avons trois grande features : les créatures, les points d'énergie et le moteur.

Dans la feature créature qui est obligatoire, on va donner le maximum de détails concernant tous les aspects des créatures (largeur du champ de vision, distance de vue, nombres, comportement, déplacement, couleurs).

Ensuite nous avons la feature points d'énergie qui est facultative, encore une fois on donne le maximum de détails sur les points d'énergie (taille, nombre).

Et finalement la feature Engine, qui donnera des infos sur l'exécution. Soit nous avons un déroulement en temps réel, soit nous avons un snapshot. Seul la feature visual est obligatoire car elle donne le type d'exécution, ensuite si nous sommes en temps réel, on peut forcer la valeur de la durée du Thread.sleep() pour avoir une exécution plus ou moins rapide, et si nous sommes en snapshot, alors nous mettons le Thread.sleep() à 0 et on choisit la durée de la simulation. Toutes les features qui ont un lien direct avec une variable dans le code sont de la forme "clé/valeur" afin de faciliter l'implémentation du lien entre FM et code. De plus tous le FM est écrit en anglais, d'une part pour faciliter le pont entre les deux et empêcher deux features d'avoir le même nom et d'autre part car tous notre code est écrit en anglais.

Domaine :

Le feature model de domaine va essayer d'être compréhensible par quelqu'un qui doit utiliser le simulateur sans en connaitre les détails techniques. Ainsi nous avons toujours trois grande features mais plus exactement les mêmes : les créatures, l'affichage et les démos.

Dans la feature créature, nous savons simplifier le tout en ne mettant que le nombre et des types de créatures qui sont des comportements et des déplacements afin de ne sortir du simulateur que des configurations comportement plus déplacement intéressant.

La feature affichage nous permet de choisir comme dans l'implémentation les mêmes choix mais compréhensibles par l'utilisateur et non seulement le développeur.

Et pour finir la feature démo, qui permet de choisir des démos soit paramétrable, soit non paramétrable. Les choix sont aussi bien fixés dans le FM de domaine que d'implémentation.

L’agrégat :

L'agrégat des deux FM se fait sans soucis en liant principalement toutes les infos correspondantes (nombre de créature d'un FM a l'autre, type de créature a un comportement et un déplacement, …), nous avons ensuite des démos qui vont uniquement fixer un environnement avec plus ou moins de points d'énergie et des tailles variable.

Nous obtenons bien à la fin trois comportement mort, ce qui était attendu vu qu’ils ne sont utilisés par aucun type de créature et aucune démo.

Le pont :

Le pont entre le feature model et le code de la simulation se fait grâce à la classe ConfigHandler.

On récupère le fichier fml qu'on évalue. On évalue ensuite un fichier de configuration se trouvant dans le dossier Config/. Les fonctionnalités sélectionnées grâce à cette configuration sont parsées.

Le parsing se fait seulement sur les fonctionnalités ayant un "_". En effet, la partie avant de l'underscore est une clé contenue initialement dans un dictionnaire et la partie après l'underscore est la valeur de la clé en question.

Ce dictionnaire contient un ensemble de clé / valeur initialement définies ce qui permet une configuration par défaut.

Une fois ces étapes terminées, le dictionnaire mis à jour est transmis au Launcher qui crée la simulation correspondante aux fonctionnalités sélectionnées.

Par exemple :

Si on veut appliquer la fonctionnalité "nombre de créature", on fait :

```
launcher.nombreCreatures(dictionnaire.get("CHC"));
```

Rajout dans le code :

-Snapshot : *permet de n'afficher que l'état de la simulation après n ticks.*

Rajout d'un booléen isSnapshot dans le Simulator.

Tant qu'on n'arrive pas à n ticks on n'affiche pas la simulation.

-Duration : *nombre de ticks avant de prendre un snapshot.*

Rajout d'un nombre de ticks max dans le Simulator.

On arrête la simulation lorsque les ticks actuels atteignent le nombre de ticks max.

-Affichage des valeurs de la configuration dans l'interface : il *n'y a plus de slider ou de choix à faire.*