Explications et justifications des choix de conceptions

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

Table des matières

1. Informations générales:

Nous avons utilisé la correction du TP5 comme point de départ, et on utilise le JDK 1.7.

2. 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 « –DscenarioTest= » suivi d'un chiffre.

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

Scénarios d'utilisation :

0 : exécution normal

1 : Le plus simple : Une seul créature stupide, pas de point d'énergie, déplacement Torus

2 : Immortelle : 20 créatures avec un comportement émergent, déplacement Torus, 15 points d'énergie de taille 50

3 : Changement de comportement pour survivre : 5 créatures avec un comportement émergent puis de recherche d'énergie, déplacement Torus, 3 points d'énergie de taille 60

4 : Piège : 15 créatures avec un comportement émergent qui devienne prédateur, déplacement Bouncing, pas de points d'énergie

5 : Chaotique : 25 créatures avec un comportement aléatoire, déplacement Bouncing, 5 points d'énergie de taille 25

3. Changement dans le package créatures :

Dissociation comportement, déplacement :

Le comportement, qui est le fait que la créature modifie sa vitesse et sa direction grâce à son environnement, ainsi que le déplacement ont été dissocié de la créature.

L'utilisation d'un patron de conception Strategy nous a semblé le plus propice, car en donnant la créature en argument on peut faire tous les calculs nécessaires soit au déplacement, soit au comportement. Ainsi maintenant une créature prend à sa création une stratégie de déplacement et une de comportement (toutes les créatures partagent la même) et c'est l'appel à la fonction setNextDirectionAndSpeed ou setNextPosition qui fera les changements sur la créature. La méthode setNextDirectionAndSpeed appelle la fonction move (qui appelle elle-même le setNextPosition de la stratégie de déplacement) afin de laisser le choix au comportement de se déplacer ou non (sans passer par une vitesse nulle ou d'autres options plus contraignantes).

Nous avons choisi le patron de conception Strategy car elle répond bien à l'intention que nous avions qui été que les comportements, mouvements soit interchangeable pour une créature et qu'ils appartiennent tous à une même classe.

Comportements et déplacements sont des plugins :

Comme les comportements et les déplacements sont des plugins, ils seront manipulés par leurs interfaces respectives IStrategyBehavior et IStrategieMovement.

Ajout d'une classe EnergySource :

Cette classe représente les points d'énergie qui vont redonner de la vie aux créatures quand elle passe dessus. EnergySource implémente l'interface IDrawable et c'est aux créatures de se redonner des points de vie (en regardant si elles sont bien sur un point d'énergie).

Ajout du système de vie :

Le système de gestion de vie se fait directement dans ComposableCreature. La vie de la créature est représentée graphiquement par son FOV. Lorsqu'une créature passe sur un point d'énergie elle

gagne un certain nombre de point de vie défini statiquement. Si la vie d'une créature est nulle, celle-ci meurt et on l'enlève de la simulation. Si une créature reste trop longtemps sur un point d'énergie, elle brule et cet effet est graphiquement représenté par une image de flamme.

Comportement recherche points d'énergie :

Le comportement de recherche de points d'energie se fait dans EnergyBehavior. On utilise quasiment la même implémentation que le creatureAround de la EmergingBehavior sauf qu'à la place de filtrer sur les créatures on filtre sur les points d'énergie. Lorsqu'une créature voit un point d'énergie il est rajouté à la liste. Si plusieurs points d'énergie sont vus par la créature on choisit par défaut la première de la liste. On calcule la direction entre le point d'énergie et la créature. Celle-ci devient la direction de la créature et on fait un move par la suite. Si la créature n'a pas vue de point d'énergie elle fait un applyNoise puis un move.

Suppression des SmartCreature, BouncingCreature et de l'AbstractCreature :

Maintenant que les créatures prennent des stratégies qui définissent leur comportement et déplacement, une seule classe de créature est suffisante (nommé ComposableCreature). L'AbstractCreature n'est plus nécessaire et tous les appels dans la plupart des fonctions qui utilisé l'interface ICreature car les créatures étaient des plugins sont remplacé par CreatureComposable.

ComposableCreature :

La classe ComposableCreature prend toutes les méthodes de AbstractCreature et en rajoute quelque unes, des fonctions et paramètres pour gérer les gains et perte de vie, une fonction act() qui appelle le setNextDirectionAndSpeed de la stratégie de comportement ainsi que la fonction gainOrLoseHealth() et une fonction move() qui appelle le setNextDirection.

CompositeBehavior :

Les classes de comportement composite vont à l'instanciation regardé si toutes les classes qu'elles utilisent sont présentes, si elles ne le sont pas une fenêtre pop pour informer l'utilisateur de mettre le ou les comportements manquant. Un comportement composite peut utiliser plusieurs comportements composites.

4. Changement dans le package plug.creatures :

PluginFactory pour le mouvement et le comportement :

On a supprimé la PluginFactory de créature et nous avons rajouté une FactoryPlugin pour le comportement, mouvement, et les couleurs qui sont des singletons.

La PluginFactory pour les couleurs et les comportements gardent la même architecture avec l'utilisation des constructeurs dans un map, alors que pour le mouvement on instancie les stratégies et on les met dans le map. On utilise toujours les constructeurs pour les couleurs et les comportements pour faciliter leurs compositions alors que les déplacements sont le moins propice au composition donc on utilise l'instance.