Module de chargement d'intelligences artificielles

Développeur: Cédric Bérenger

# Présentation

Le but du module d'intelligences artificielles est de charger des scripts utilisateurs et d'en déduire les actions que les tanks effectuerons pour les communiquer au module qui gère la physique du jeu.

Le cahier des charges spécifiant que les scripts doivent pouvoir être écrit dans le langage python, et comme l'équipe de développement et moi même sommes peu familiarisé avec ce langage, nous avons choisi d'effectuer la majorité de la programmation en langage Java, puis d'interfacer les scripts pythons utilisateurs avec ce langage.

# Solutions techniques choisies

De manière à interfacer facilement les scripts python utilisateurs avec le langage java, nous avons choisi d'utiliser la technologie Jython, un interprète python open source codée en java, dont la force réside dans le fait que les scripts pythons exécutés peuvent avoir accès à des objets et des méthodes java et inversement. De plus, Jython s'avère compatible avec les threads Java, il est donc possible d’exécuter en parallèle plusieurs instances de l'interpréteur Jython, ce qui nous est nécessaire, étant donnée que nous voulons piloter plusieurs tanks en même temps et donc exécuter plusieurs scripts python en même temps.

Note : Il existe aussi une spécification de java 6, la JSR 223, qui a abouti au framework “Scripting for the Java Platform”, qui permet au langage java d'exécuter des scripts dans divers langages de programmation (python, javascript etc.) via un interpréteur à inclure. Le fonctionnement de ce framework est très similaire à celui de Jython. Pour plus d'informations sur ces technologies, voir l'article correspondant à l'intégration de Jython et Java sur la documentation jythonbook: [http://www.jython.org/jythonbook/en/1.0/JythonAndJavaIntegration.html#jsr-223](http://www.jython.org/jythonbook/en/1.0/JythonAndJavaIntegration.html" \l "jsr-223)

# Architecture

Le module se décompose en 3 classes principales :

* La classe Intelligence, qui charge les scripts pythons utilisateurs, puis les exécutent sur un thread dédié avec l'interpréteur de python pour java Jypthon.
* La classe TankRemote, qui génère des « télécommandes » permettant aux scripts pythons de piloter leurs tanks respectifs.
* La classe Intelligences, qui génère les Intelligences (classe Intelligence), les met dans une liste et les supervises.

# Fonctionnement

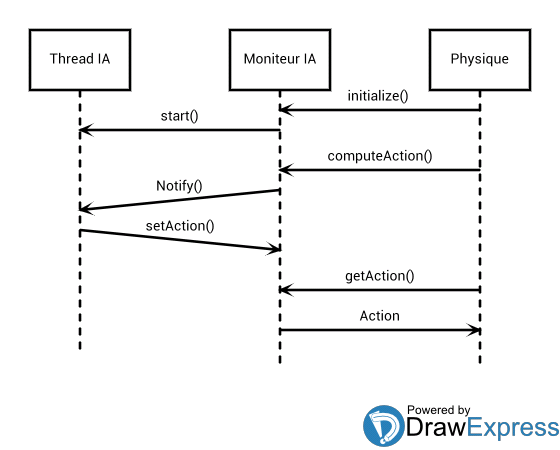
Le fonctionnement du module est le suivant :

* Chaque tank possède une intelligence programmée par un utilisateur.
* Lorsque le tank est crée, un nouvel interpréteur jython est instancié sur un nouveau thread (classe Intelligence). Cet interpréteur charge alors un script utilisateur passé en paramètre, puis il exécute le script.
* Le script utilisateur Jython / Python à accès a un objet « télécommande » (classe TankRemote) qui lui permet de choisir la prochaine action du tank. Pour faire son choix, le script a aussi accès, via cette télécommande, aux différents capteurs du tank.
* Une fois l'action choisi sur la télécommande, celle-ci se verrouille et le thread hébergeant le script utilisateur s'endort.
* La partie physique s’exécute en parallèle sur un thread différent. Lorsque celle-ci, pour mettre à jour le jeu, a besoin de connaître les intentions de l'ia d'un tank, celle-ci appelle la méthode computeAction() de l'objet ia. Cela a pour effet de déverrouiller la télécommande du tank et de réveiller son ia.

Ainsi, l'IA utilisateur, c'est à dire le script python/jython peut alors continuer son exécution et calculer une nouvelle action à effectuer, ce qui peut prendre un certain temps.

* Comme la partie physique s'éxécute en parallèle, celle-ci doit laisser le temps au script de choisir son action. Si le script met trop de temps à répondre, elle peut considérer le thread comme fautif et continuer mal-grès tout le cours du jeu. Cela permet de ne pas rester bloqué en cas de boucle infini par exemple.
* Une fois la nouvelle action choisi, la télécommande se reverrouille et ainsi de suite.

Voici un petit diagramme de séquence :



Test unitaire de la classe

Avant d'intégrer le module d'intelligences dans le reste du projet, ce qui consistera principalement à l'interfacer avec la partie physique, il convient de vérifier, de manière indépendante, que les classes de ce module fonctionnent correctement, cela à fin de faciliter la localisation de bugs lors de l'intégration en excluant des recherche la partie intelligence (puisque déjà vérifiée).

## Procédure de test

Pour vérifier le bon fonctionnement du module d’intelligence artificielle, j'ai simulé un cas concret d'utilisation des classes mettant un œuvre plusieurs tanks (et donc plusieurs IA).

Puisque je ne dispose pas de la partie physique pour mes tests, j'ai simulé manuellement les appels de méthodes que pourra faire la partie physique sur les classes du module en essayant le plus de cas possibles.

Pour ce faire, j'ai crée plusieurs scripts python / jython, un pour chaque tank ainsi qu'une classe de tests.

Dans le main de cette classe, j'instancie une nouvelle liste d'intelligence (un objet de la classe Intelligences) , je crée et j'initialise plusieurs intelligences (4 par exemple), et je lance plusieurs appels aux différentes méthodes publiques des ia, notamment les méthode computeAction() et getAction() qui constituent le cœur du module.

J'affiche alors les différents résultats des appels aux méthodes et vérifie la cohérence des résultats.

## Résultats des tests

Après avoir répété plusieurs fois la procédure de test et essayé plusieurs variantes, le module d'intelligence artificiel semble stable et prêt pour son intégration avec la partie physique du jeu.

# Problèmes rencontrés

Certains problèmes sont apparus lors de l'utilisation de threads java avec l'interpréteur Jython. En effet, Jython (tout comme les implémentations de la JSR 223) se comporte comme une boîte noire, il est ainsi difficile de savoir comment ce dernier va ce comporté associé avec les thread java : est-il thread-safe ? Monothreadé ?

Il s'avère, que Jython est compatible avec les thread java, dans le sens où, depuis un script python, il est possible d'accéder à un moniteur et d'endormir le script tout comme il est possible de réveillé le script après un appel a une méthode. Cependant, Jython masque certaines interruptions de thread java, ce qui rend difficile voir impossible l'arrêt d'un script python qui bouclerai à l'infini.

# Conclusion

Le module de chargement d'intelligence artificielle permet le chargement dynamique des scripts python utilisateurs puis leurs exécution grâce à l'interprète python codée en java Jython. Chaque script python tourne en arrière plan sur un thread dédié, en se tenant prêt a divulguer l'action que devra effectué le tank à la partie physique.

Les nombreux tests effectués sur ce module, notamment en simulant les appels aux méthodes par la partie physique, nous permette d'avoir assez confiance en son bon fonctionnement pour dans un premier temps l'exclure lors de la recherche de bugs suite aux tests d'intégrations.

Finalement, même si l'utilisation de l'interpréteur Jython pose quelques problèmes en pratique, notamment en ce qui concerne l'arret des thread l'hébergeant, il constitue une solution idéale pour introduire le python comme langage de scripting à l'intérieur d'une application développée en Java. Sont utilisation nous a permis d'être plus productif, en dispensant l'équipe de l'utilisation d'un langage quelle maîtrise moins bien.