MoSiMa - TP n°1

Inférence et problème de l'ancrage

1 Environnement du Duel

Les agents, au sens de programmes informatiques, sont de plus en plus souvent amenés à évoluer au sein d'environnements partiellement inconnus ou susceptibles d'évoluer en cours d'utilisation (dynamiques). Garantir l'atteinte d'un objectif donné – voire l'amélioration des performances des agents – dans un tel contexte nécessite de pouvoir faire évoluer les comportements des agents au cours du temps. Pour des agents adaptatifs, cela peut-être réalisé en faisant évoluer leurs connaissances.

Pour ce faire, deux grandes approches sont généralement considérées :

- Mettre "manuellement" à jours celles-ci, ce qui nécessite de connaître et caractériser les changements survenant dans l'environnement de l'agent, et de pouvoir communiquer avec ce dernier afin de lui transmettre les mises à jours.
- Doter les agents de capacités d'adaptation, par le biais de processus de raisonnement, de mécanismes d'apprentissage, d'abstraction et de généralisation.

Chacune de ces approches dispose d'avantages et d'inconvénients qui les rendent adaptés à certains contextes applicatifs, et la mise à jours "manuelle" est aujourd'hui la solution de référence dans le secteur industriel. Néanmoins la complexité des applications développées allant croissant, les volumes de données à traiter et l'intrication des différents composants ne permettent plus aux humains d'être suffisamment réactifs. La mise en œuvre de systèmes adaptatifs devient dès lors nécessaire.

Dans le cadre de ce cours, nous nous intéresserons ainsi aux architectures et mécanismes permettant de doter les agents d'une certaine robustesse et "flexibilité" face aux changements survenant au sein de l'environnement dans lequel ils évoluent.

Comme vu en cours, nous utiliserons comme exemple illustratif le cadre d'un duel entre deux agents évoluant sur différentes cartes (une nouvelle carte constituant un nouvel environnement pour nos agents). Le comportement de l'un de ces agents sera de votre responsabilité, et le second piloté par une IA simpliste. En utilisant l'architecture fournie, vous devrez proposer et mettre en œuvre une stratégie et un mécanisme permettant à votre agent d'atteindre son objectif (survivre et battre l'adversaire) de manière "efficace" indépendamment de la topologie de la carte considérée.

Un tel agent doit donc au minimum :

1. Disposer d'une représentation de ce qu'est une situation "intéressante" pour la victoire qui soit relativement indépendante d'un fichier de carte donné.
2. Détecter via (notamment) son moteur d'inférence, si sa position actuelle lui permet d'atteindre son objectif.
3. Rechercher, si tel n'est pas le cas, les situations "intéressantes" perceptibles dans son environnement (en comparant la situation idéale avec les différentes situations qu'il peut espérer trouver), et s'arranger pour s'y trouver.

Plusieurs solutions permettent à un agent de construire la représentation d'une situation "intéressante".

2 Prise en main de l'environnement

Démarrage de la machine virtuelle

— Sur les machines de la PPTI (en 502 uniquement) [1]

1. Ouvrez un terminal et lancez : **Vbox ACTC_2019**
   Il est possible que certains d'entre-vous se heurtent à un message d'erreur en lançant la VM : `QT FATAL : QXcbConnection : Could not connect to display : 0 /usr/bin/xauth : timeout is locking`

   Pour y remédier il suffit de faire un `chmod 755 .Xauthority` ou sinon un `chmod 755 .` à la racine du compte.
2. Lancez eclipse depuis le bureau de la vm

— Sur votre propre machine

1. Faites moi signe et récupérez la machine virtuelle fournie sur clé (Linux Mint 64bits, 4.5Go). Elle est configurée pour consommer 3Go de ram, 2 procs, et l'accélération graphique. Vous pouvez revoir à la baisse ou à la hausse ses caractéristiques en fonction de la puissance de votre machine.
2. Lancez-là puis lancez Eclipse.

Root sur la machine virtuelle
Identifiant : `cmt` – Mot de passe : `mosima2018`

Arborescence du projet

Ce projet repose sur différents composants logiciels : *JmonkeyEngine (jme3)* comme moteur pour l'environnement, *Jade* comme plateforme multi-agent, *Prolog* comme mécanisme d'inférence et *Weka* pour les algorithmes d'apprentissage.

src La version dont vous disposez est structurée autour de 3 répertoires :

- env A la charge de l'environnement dans lequel vont évoluer vos agents. Sauf à vouloir modifier la physique ou les interactions avec celui-ci, vous n'aurez à intervenir ici.
- princ C'est ici que vous démarrez l'application et indiquez le nom et la classe des agents évoluant sur la carte.
- sma C'est en partie ici que vous aurez à travailler pour modifier le comportement de votre agent et obtenir des résultats satisfaisants. Vous y trouverez actuellement un agent élémentaire, qui se déplace de manière pseudo-aléatoire sur la carte, et un agent un peu plus intelligent utilisant prolog pour raisonner.
- tests Comme son nom l'indique, vous y trouverez des exemples illustratifs pour la génération de cartes, et l'appel des différentes libraires utilisées. Le tp commencera dans ce répertoire, avec l'utilisation de prolog.

ressources C'est ici que sont stockées les cartes (sous la forme de HeightMaps) et les jeux d'instruction pour le moteur d'inférence. C'est également ici que vous stockerez les différentes "situations" perçues par vos agents lorsque nous doterons ceux-ci d'un mécanisme d'apprentissage (`simus`).

1. Pour des raisons de sécurité, la PPTI interdit les transferts de fichiers host-vm, mais l'accès internet est ouvert.

Moteur d'inférence

Lisez, exécutez, et comprenez la classe `TestPrologCalls2Ways.java` que vous trouverez dans le répertoire `tests/prologTest/`, et sa classe prolog associée, disponible dans `ressources/prolog/test/fishing.pl`. Le cycle d'exécution réalisé dans l'exemple donné est un élément clé du fonctionnement général de l'architecture présenté en cours. Il est important de bien en visualiser les différentes étapes.
Le détail des API jpl (prolog et java) est disponible ici : `https://jpl7.org/JavaApiOverview`

Réalisation d'un duel

Exécutez le programme fourni en lançant la class `Principal.java` du `package src/princ/`. Ne touchez pas à votre souris ou votre clavier avant que les agents ne soient déployés sur la carte. Déplacements de la caméra :

- J/L : Déplacement latéral gauche/droit
- I/K : Déplacement latéral haut/bas
- P/M : Zoom in/out
- Bouton de souris enfoncé + mvt de souris : rotation

En fin d'exécution, l'agent "intelligent" sauvegarde la dernière situation perçue dans un fichier.

3 Inférence et ancrage - à rendre pour le 9 décembre

Le travail (réponse aux questions, code de vos agents [.pl et .java] et bases de situations) est à rendre sous la forme d'une archive et à envoyer par mail à *cedric.herpson@lip6.fr* pour le lundi 9 décembre 8h. L'objet de votre mail comme le nom de votre archive doit être : **Mosima - TP1 - Nom1 et Nom2**.

1. En vous appuyant sur le cours et l'article de Langley *et al* fourni, rappelez en un paragraphe ce qu'est le problème de l'ancrage puis quels sont les avantages et inconvénients des principales approches d'architectures cognitives existantes.
2. Étudiez et comprenez le raisonnement de l'agent Dummy et de l'agent dit "intelligent".
   (a) Quelle est la stratégie de l'agent Dummy, de combien de comportements est-elle constituée ?
   (b) Représentez la stratégie de l'agent dit "intelligent" sous la forme d'un automate.
3. Élaborez votre propre agent
   (a) En vous inspirant du comportement de l'agent "intelligent" existant, créer un nouvel agent dont le comportement le conduira à toujours privilégier lors de ses déplacements l'endroit le plus haut perçu dans son champ de vision, avec une probabilité de choisir une autre destination égale à 20%.
   (b) Modifiez votre code de façon à sauvegarder la situation courante en cas de victoire/défaite dans le répertoire ressources/learningBase/(victory/defeat)
   (c) Modifiez votre code de façon à ce que votre agent sauvegarde en plus (dans un répertoire différent) la situation courante à chaque fois qu'il est touché ou perçoit son adversaire. Vous disposerez donc d'une base de donnée de victoires et défaites et d'une base de donnée possédant les situations associées à un changement d'état (activation de la possibilité de tir, ou modification de son état de santé), cette dernière se remplissant beaucoup plus rapidement.