TP1 : Graphes et Behaviour Trees

21 juin 2019

1 Objectif des TPs

Ce TP est le premier d'une série de trois TPs dont le but final est de créer une interface pour un jeu de PacMan. Dans ce TP, vous verrez un outil qui s'appelle le Behaviour Tree. C'est un outil qui sert dans les domaines du jeu-vidéo et de la robotique pour contrôler les agents autonomes (i.e. NPC, robot, ...).
Dans le second TP, vous apprendrez à créer une interface tkinter.
Et dans le dernier TP, vous mettrez en place une interface pour un PacMan.
Et si vous êtes rapide, vous pourrez même créer des Behaviour trees pour contrôler le PacMan comme vous le voulez.

2 Rappels sur les graphes :

Les graphes sont des structures de données composée de noeuds et d'arêtes. Une arête fait le lien entre deux noeuds. On appelle enfant les noeuds auxquels on peut accéder directement depuis un noeud.

Un graphe est dit orienté lorsque ses arêtes sont orientées. C'est à dire lorsqu'elles possèdent une origine et une extrémité. Ces arêtes ne peuvent être parcourues que dans un seul sens, de l'origine à l'extrémité.

Un graphe est dit acyclique lorsqu'il ne possède pas de cycles. C'est à dire que si l'on part d'un noeud, on ne peut pas trouver de chemin qui ramène à ce noeud.

Il existe plusieurs façons de parcourir un graphe. Mais pour ce TP on se concentrera sur le parcours en profondeur. Parcourir un graphe en profondeur se fait de manière récursive et se déroule selon les règles suivantes pour chaque noeud :

— Marquer le noeud comme visité
— Effectuer l'action à réaliser sur ce noeud
— Pour tous les enfants non marqués de ce noeud effectuer les mêmes actions

3 Graphes simples :

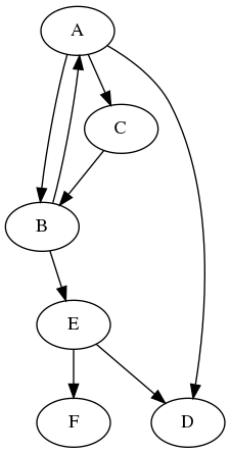
Pour ce premier exercice, vous aurez à créer une classe `Noeud` qui vous permettra de construire un graphe. Les attributs de ce `Noeud` sont :

— *nom* : Le nom du `Noeud` donné à l'initialisation
— *enfants* : Une liste des enfants de ce `Noeud`
— marque : Booléen indiquant si le `Noeud` a été visité

Les méthodes de ce `Noeud` sont :

— `__init__(self, nom)` : Permet la création d'un `Noeud` et l'initialisation de ses attributs
— ajouterEnfant(self, enfant) : Ajoute un enfant à la fin de la liste des enfants
— parcours(self) : Permet le parcours en profondeur du graphe, l'action à réaliser sera l'affichage du nom du `Noeud`

Après avoir créé la classe `Noeud`, construisez le graphe suivant en ajoutant les enfants du plus à gauche au plus à droite.



Le parcours sur ce graphe en partant de "A" devrait donner : A-B-E-F-D-C

4 Behaviour trees :

Un behaviour tree (BT) est un outil permettant de concevoir un comportement pour un agent autonome, il permet également de l'exécuter. Pour évaluer un BT, on lance un parcours du graphe depuis la racine du graphe, parcours que l'on va appeler *tick*. Chaque noeud va donc évaluer ses enfants selon des règles qui sont associées au type de noeud auquel ils appartiennent. Pour chaque évaluation, un noeud devra retourner à son père son état qui peut être Succès, Échec ou En Cours.

Les **noeuds composites**, ce sont les noeuds de parcours du graphe, ils parcourent leurs enfants selon leurs règles et retournent leur état selon l'état de leurs enfants. La plupart des noeuds évaluent leurs enfants de gauche à droite. Les noeuds composites de la bibliothèque *py-trees* sont les suivantes :

- `Sequence` : Il évalue l'enfant suivant uniquement si le précédent retourne Succès. Sinon il retourne l'état du dernier enfant. Il ne retourne Succès que lorsque tous les enfants ont réussi.
- `Selector` : Il retourne Succès dès le premier enfant qui retourne Succès. Si aucun enfant ne retourne Succès, il retourne Échec.
- `Parallel` : Il évalue tous ses enfants en même temps. Dans son comportement de base, si tous les enfants réussissent, il retourne Succès. Sinon il retourne En Cours.
- `Chooser` : Vous n'en aurez fort probablement pas besoin.

Les **noeuds d'exécution**, ce sont les noeuds qui interagissent avec le modèle, ils éxécutent une fonction du modèle. La bibliothèque *py-trees* ne proposant pas d'implémentation de ces noeuds, utilisez le noeuds Action se trouvant dans *toolkit.py*. Ce noeud permet le lancement d'une fonction avec au plus un argument. Si la fonction retourne `True`, le noeud retourne Succès. Si la fonction retourne `False`, le noeud retourne Échec. Si la fonction retourne `None`, la fonction retourne En Cours.

Il existe également les **noeuds décorateurs**, ce sont des noeuds ne prenant qu'un seul enfant, et qui changent l'état de retour de cet enfant selon des règles qui ont été définies pour ce décorateur. Par exemple, la bibliothèque *py-trees* propose le décorateur `Inverter` qui inverse l'état entre Succès et Échec.

Pour tout renseignement supplémentaire, la documentation de *py-trees* se trouve à l'adresse suivante : `https://py-trees.readthedocs.io`

Pour vous familiariser avec la bibliothèque, codez le comportement suivant avec l'aide des fichiers *trameBT.py* et *robot.py* :