TRAVAUX DIRIGES DE SIMULATION
ENSTA BRETAGNE – TD6
ANNEE 2019-2020

Enseignant : Olivier VERRON (simuenstabretagne@gmail.com)

1. PREAMBULE

Cette partie du TD va vous préparer à réaliser le bureau d'étude.

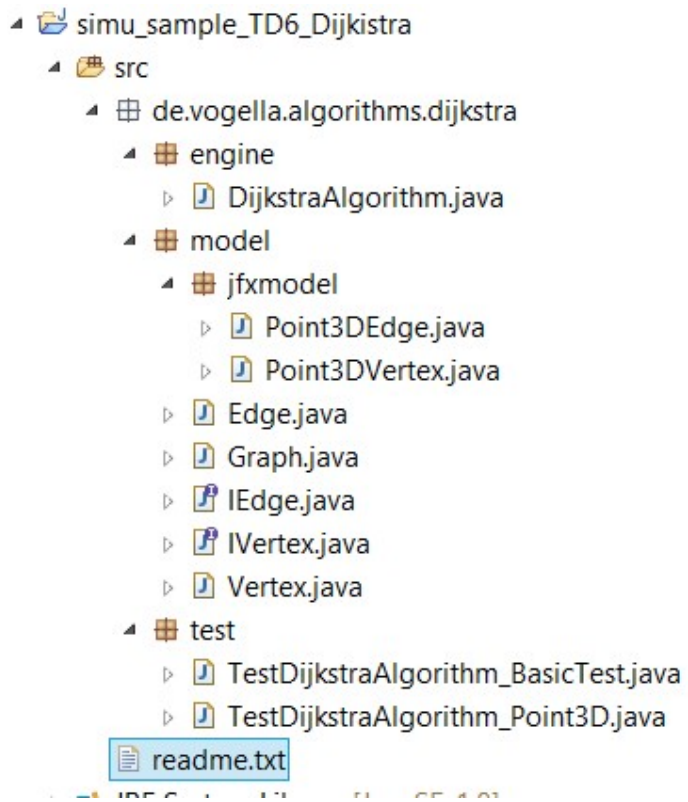
2. OBJET DE L'EXERCICE

- Manipuler une bibliothèque de calcul de plus court chemin
- Comprendre le fonctionnement de quelques bibliothèques fournies pour la réalisation du futur BE
- Utiliser la bibliothèque pour réaliser une tâche un peu plus complexe

3. ALGORITHME DE PLUS COURT CHEMIN

3.1 Préparation de l'espace de travail

Importer le projet `simu_sample_TD6_dijkistra.zip` situé sur Moodle.



Le projet a été trouvé sur Internet et légèrement adapté.
Regarder le contenu du package Test.
Basic Test est l'utilisation brute de l'algorithme.

Point3D exploite la notion de JavaFX pour calculer le poids des liens entre points.
L'algorithme tel qu'implémenté repose sur l'implémentation de deux interfaces IEdge et IVertex et d'un moteur parcourant un graphe.

3.2 Fonctionnement

Vus devez avoir vu la partie algorithme en recherche opérationnelle. L'objet ici est juste de comprendre comment utiliser l'algorithme.

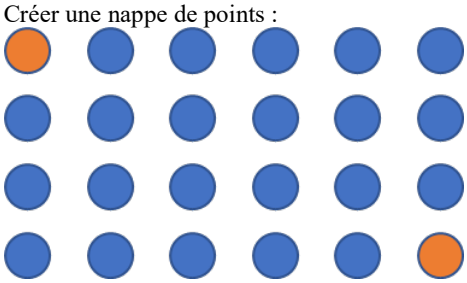
Ci-dessous l'algorithme créer des liens entre divers points et les références au passage dans deux listes (edge = lien entre deux neuds.
La fonction `addLane` crée un lien dans un sens ET dans l'autre.
On lance l'algorithme avec `Exécute` à qui on passe le point de départ.
Puis par le `getPath` on récupère le chemin pour atteindre une cible.

```
nodes = new ArrayList<IVertex>();
    edges = new ArrayList<IEdge>();

    addLane(new Point3DVertex(0,0,0), new Point3DVertex(0,1,0));
    addLane(new Point3DVertex(0,1,0), new Point3DVertex(1,1,0));
    addLane(new Point3DVertex(1,1,0), new Point3DVertex(1,2,0));
    addLane(new Point3DVertex(1,2,0), new Point3DVertex(2,2,0));
    addLane(new Point3DVertex(2,2,0), new Point3DVertex(4,4,0));

//    addLane(new Point3DVertex(0,0,0), new Point3DVertex(4,4,0));

    // Lets check from location Loc_1 to Loc_10
```

```
Graph graph = new Graph(nodes, edges);
DijkstraAlgorithm dijkstra = new DijkstraAlgorithm(graph);
dijkstra.execute(new Point3DVertex(0,0,0));
LinkedList<IVertex> path = dijkstra.getPath(new Point3DVertex(4,4,0));

for (IVertex vertex : path) {
  System.out.println(vertex);
}
```

3.3 Exercice

Créer une nappe de points :



Trouver le plus court chemin entre les deux….

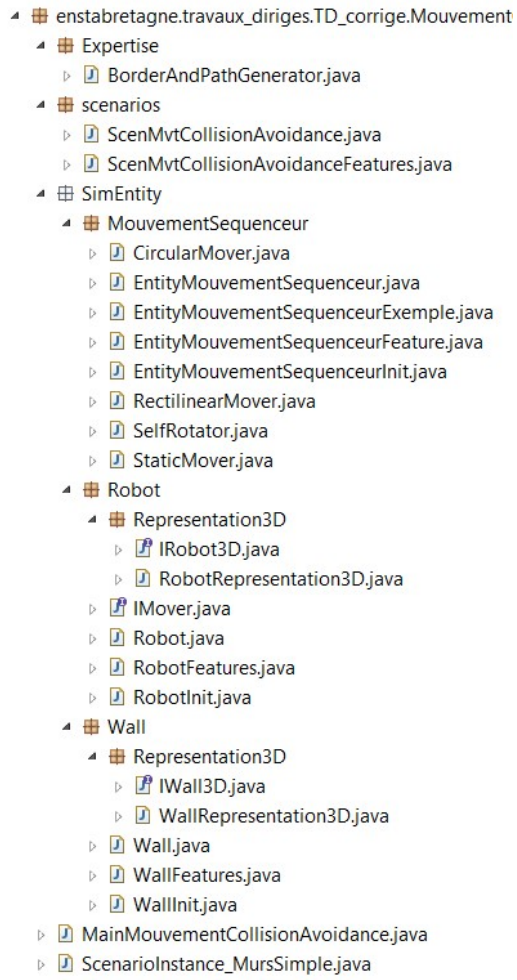
4. MISE EN ŒUVRE D'UN ROBOT EN 3D

Cette partie va vous apprendre à :

- Créer un mur
- Donner un mouvement continue de manière plus élégante que ce qui a été fait en TD4
- Traiter le problème de l'intervisibilité

Récupérer le projet `simu_sample_TD6_Robot pour BE.zip`

4.1 Vue du projet



Il vous fournit en expertise une bibliothèque (`BorderAndPathGenerator`) permettant de réaliser des opérations en 3D. Les fonctions sont commentées et les exemples fournis les utilisent.

Le mur (Wall) est une entité de simulation. Elle possède au niveau des features un type. Il est proposé de considérer du mobilier par exemple comme un Wall d'un type 1. Les vrais murs sont pris comme de type 2. Un énuméré serait sans doute plus judicieux.
Il se caractérise en Init par un chemin de point successif et un algorithme situé dans WallInit (`generateMurs`) transforme ces points en murs.

Le Robot peut être gentil ou méchant. Il a une destination de déplacement, une origine ET une orientation.
Seul le gentil a des capacités de déplacement par une sous entité de simulation le séquenceur et de détection au début de ce TD.
Il a une capacité de détection simple :

- Il détecte les murs
- Il détecte les objets

Il peut rechercher le robot ennemi d'abord sans tenir compte de l'intervisibilité.

Le robot au démarrage se demande s'il voit vraiment en intervisibilité la table d'où il se trouve avec `canSeeTable()`

Puis ne voyant pas, il doit se déplacer.

Il active alors le séquenceur de mouvement qui sera présenté au § suivant qui va le tourner dans la bonne direction puis l'aider à se déplacer.
Lorsque le séquenceur a finit le job, le robot a bougé et peut retester avec succès la visibilité de la table.

4.2 Le séquenceur

Le séquenceur s'appuie sur une astuce : l'extrapolation linéaire.
On fournit 4 comportements d'extrapolation: les movers.

- le statique qui se contente d'avoir une position et une orientation fixe
- le rectilinéaire qui peut à partir d'une position initiale, d'une vistesse initiale extrapoler une position dans le temps
- le circulaire qui peut à partir d'une position initiale, d'une orientation initiale, d'une vitesse de se déplacer selon un cercle dpour atteindre une position
- la rotation sur soit qui permet à partir d'une position et d'une orientation initiale, d'atteindre un angle de rotation déterminé

Le séquenceur chapeau offre toujours la même interface au robot. Quand le robot fait getposition() sur le séquenceur en fait le séquenceur fait un getPosition(getCurrentTime()) sur le mover qui a dû être correctement initialisé au préalable.
Le séquenceur enchaine en transparence pour le robot les actions.

L'enchainement est réalisé par le fait qu'on sait résoudre le temps mis pour atteindre la cible via la fonction `getDurationToReach()`. On poste un événement sur la fin d'opération de rotation ou de déplacement. Cet événement correspond à la fin du mouvement et on peut alors déclencher une autre action.

Dans le cas fournit par défaut, il se tourne vers une cible, puis se déplace vers la cible.

- Pour le faire aller ailleurs, il faudra réinitialiser soit `selfrotator` sur la dernière position pour se mettre dans la bonne direction puis réinitialiser un rectilinear pour aller ailleurs, sans oublier de poster les événements de fin.

Cette astuce permet au visuel d'appeler à 25Hz la position sans pour autant qu'un seul événement ou changement d'état ne se produise réellement !

C'est pourquoi on a un grand différentiel de vitesse entre un moniteur visuel et non visuel, ce dernier ne faisant pas d'appels intermédiaires aux fonctions !

4.3 Les services 3D

Afin de percevoir ce que perçoit le robot il faut souvent étendre l'interface Robot3D entre le visuel et l'entité Robot. Dans l'exemple fournit on a mis **getLineOfSight()** qui permet de vérifier ce que le dernier appel d'intervisibilité réalisé et voir avec nos yeux si le résultat est bien le bon !

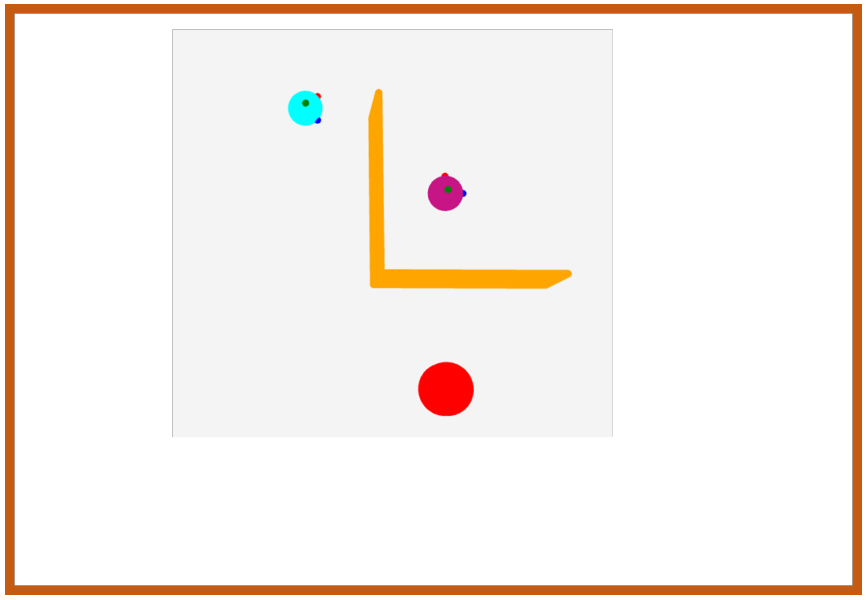
Notez qu'au niveau de la représentation 3D du robot le ligneOfSite est ajouté aux children de myworld et non du robot ! Sino la lineofsight suivrait le mouvement du robot !

5. EXERCICES PREPARATOIRES AU BE (NON CORRIGE)

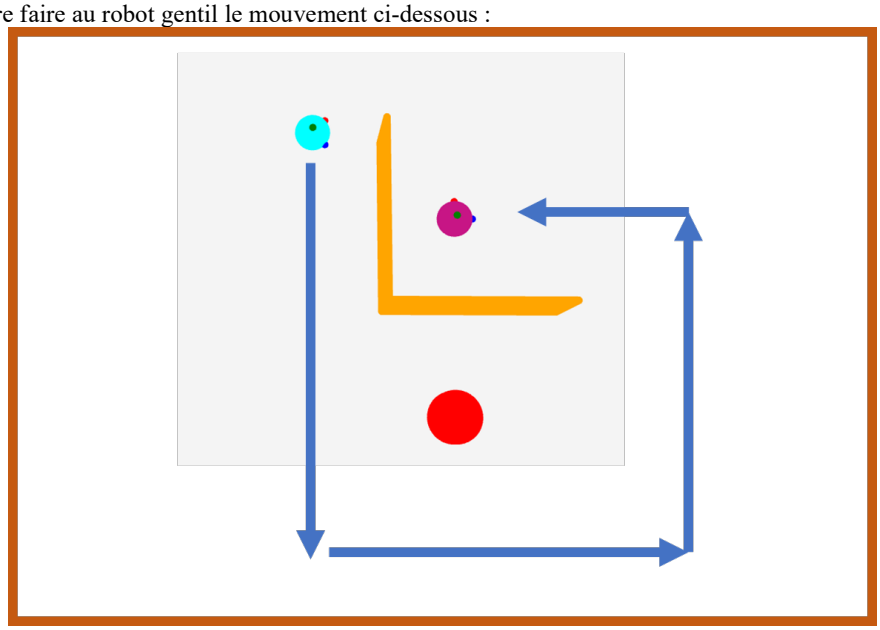
Pour réussir le BE il faudra savoir piocher dans les différents TD et corrigés fournis.

Ici il vous est proposé :

1. Bien regarder le code, modifiez les paramètres d'initialisation du scénario MurSimple
2. Premier exercice, créer un mur rectangulaire Orange comme ci-dessous autour de la scène fournie



3. Faire tester au robot gentil s'il est en intervisibilité avec le robot violet
4. Faire faire au robot gentil le mouvement ci-dessous :



5. Faire tester de nouveau l'intervisibilité
6. Enregistrez le temps mis